Project: **PICX** Datum: 8.12.2019
Name: Duy Khoi Nguyen
Matr.-Nr.: 630305

Hochschule Flensburg
University of Applied Sciences

Angewandte Informatik

Webprogrammierung

Hausarbeit WS19

Von

Duy Khoi Nguyen

Mtrn. 630305

Inhalt

1. Quellcode Go

```
package main

import (
	"bytes"
	"errors"
	"fmt"
	"image"
	"image/color"
	"image/draw"
	"image/png"
	"io"
	"math"
	"math/rand"
	"net/http"
	"sort"
	"strconv"
	"strings"
	"text/template"
	"time"

	"github.com/disintegration/imaging"
	"github.com/globalsign/mgo"
	"github.com/globalsign/mgo/bson"
)

type UserCredential struct {
	Username string
	Password string
	Albums   []string `bson:"albums"`
```

```
}

type UserCredential2 struct {
        Id      bson.ObjectId `bson:"_id"`
        Username string
        Password string
        Albums   []string `bson:"albums"`
}

type LoginSignInFeedback struct {
        Feedback string
        Color    string
}

type PoolNamesStrc struct {
        PoolNames     []string
        PoolFeedback  string
        FeedColor     string
        PictureCount  []string
        ShowKachelSize []string
        Kachelsizes   [avaibleSizeNumb]int //der hier ist nur um das selectfeld zu generieren
}

type MosaicStrc struct {
        Albums        []string
        PoolNames     []string
        PoolFeedback  string
        FeedColor     string
        PictureCount  []string
        ShowKachelSize []string
```

```
	AfterSource    string `bson:"aftersource"`
	BeforeSource   string `bson:"beforesource"`
}

type fileTemplateStrc struct {
	ID         bson.ObjectId `bson:"_id"`
	Filename    string       `bson:"filename"`
	Length      int32        `bson:"length"`
	UploadDate  time.Time     `bson:"uploadDate"`
	Source      string       `bson:"source"`
	Metadata    Metadatas2    `bson:"metadata"`
	Aufloesung  string       `bson:"aufloesung"`
	IDHexstring string
	AuflosungX  string
	AuflosungY  string
	DbFileDir   string
}

type ImagesStrc struct {
	PageSite       string //zur unterscheidung BaseMotifs und Mosaicgallery weil beide das selbe template benutzen
	Poolname       string
	CollectionName string
	Images         []fileTemplateStrc `bson:"images"`
	Albums         []string
}

type Metadatas2 struct {
	MiddleColorVec Vector3D `bson:"middleVector"` //MiddleColorVec besteht as r g b
	Brightness     float64  `bson:"brightness"`   //Helligkeit, die länge von MiddleColor
```

```
        Aufloesung    string   `bson:"aufloesung"`
        Album         string
}

type Metadatas struct {
        //MiddleColor    color.Color `bson:"middleColor"`  //MiddleColor besteht as r g b a
        MiddleColorVec Vector3D `bson:"middleVector"` //MiddleColorVec besteht as r g b
        Brightness     float64  `bson:"brightness"`   //Helligkeit, die länge von MiddleColor
        Kachelsize     string   `bson:"kachelsize"`
        Aufloesung     string   `bson:"aufloesung"`
}

type Vector3D struct {
        X, Y, Z uint8
}

type Vector3Df struct {
        X, Y, Z float64
}

type Kachelstrct struct {
        Brightness  float64
        FileName    string
        Farbabstand float64
        ID          bson.ObjectId `bson:"_id"`
}

type Graphstrct struct {
        Poolname    string
        AvgRGB      Vector3Df
```

```
	AvgBrightness float64

	AvgDrawCoord  Vector3Df

}


type BrightnessSort []Kachelstrct

type FarbabstandSort []Kachelstrct

type UploadTimeSort []fileTemplateStrc


const avaibleSizeNumb = 6


var kachelsizes = [avaibleSizeNumb]int{5, 10, 15, 20, 25, 30}


//keine Enrükungen oder lerzeichen in FeedbackString

var feedbackString = `

{{if .Feedback}}

<div id="feedbackID" style="color:{{.Color}};">{{.Feedback}}</div>

{{end}}

`

var wholeGalleryPage = `

<!DOCTYPE html>

<html>

	<head>

   <link rel="stylesheet" href="CSS_FONTS/picxStyle.css">

   <script src="JS/PICX.js"></script>

   <meta charset="UTF-8">

   <meta name="viewport" content="width=device-width, initial-scale=1.0">

   <meta http-equiv="X-UA-Compatible" content="ie=edge">

   <title>PICX Hausarbeit Webprogrammierung</title>

	</head>

	<body>
```

```
  <div class="box">

    <header class="row header">

      <div class="logo links">PICX</div>

      <nav class="center">

        <ul class="nav_center">

                                    <li id="galleryID"><a id="AGallery"
href="/gallery">GALLERY</a></li>

                                    <li id="baseMotifsID"><a id="ABaseMotifs"
href="/baseMotive">BASIC MOTIVES</a></li>

                                    <li id="baseCreationID"><a id="AMosaic"
href="/mosaic">MOSAIC CREATION</a></li>

                                    <li id="imagePoolID"><a id="AImgPool"
href="/imgPool">IMAGE POOL</a></li>

        </ul>

      </nav>

      <div class="rechts directionColumn" id="rechts">

        <li><a class="logoutA" id="profile"><img id="logout" src="Icons/profile.png"> &#9660;
</a></li>

        <ul id="submenu1">

          <li><a id="settingsID" href="/settings"><img class="submenuIMAGE"
src="Icons/settings.png">setting</a></li>

          <li><a id="logoutID" href="/logout"><img class="submenuIMAGE"
src="Icons/logout.png">logout</a></li>

        </ul>

      </div>

    </header>

    <div class="row content">

                        <div class="siteTitle mosaicBasicTitle" id="{{.PageSite}}">{{.PageSite}}</div>



                        <div id="selectAlbumDiv">

                        <select name="album" id="albumSelection">

                                <option selected disabled hidden>Choose Album</option>
                                <option value="All Images">All Images</option>

                                {{range $i, $album := .Albums }}

                                        <option value="{{$album}}">{{$album}}</option>

                                {{end}}

                        </select>

                        <span class="dropdown" id="deleteDropdownID">

                                <span class="dropdownOption"
id="dropdownOption">&vellip;</span>

                                <div id="dropdownDelete" class="dropdownDelete">

                                        <span id="deleteAlbum" >Delete Album</span>

                                </div>

                        </span>

                </div>

                {{if .Images}}

                        <div class="grid-containerGallery" id="gridBoxGallery">

                        {{range $i, $img := .Images }}

         <div class="grid-item">

           <img class="grid-img" id="{{$img.DbFileDir}}" src="{{$img.Source}}">

           <span class="overlay">

                                                <a
href="/downloadMosaicOrBasic?download={{$img.DbFileDir}}"><img class="overlayDownload"
id="{{$img.DbFileDir}}" src="Icons/download2.png"></a>


                <img class="overlayInfo" src="Icons/information.png"

                title="Bildgröße: {{$img.Length}} Bytes &#10;UploadDate: {{$img.UploadDate}}
&#10;Name: {{$img.Filename}} {{if $img.Aufloesung}}&#10;Aufloesung: {{$img.Aufloesung}} {{end}}
{{if $img.Metadata.Brightness}}&#10;Helligkeit: {{$img.Metadata.Brightness}} {{end}}

 {{if $img.Metadata.Album}}&#10;Album: {{$img.Metadata.Album}} {{end}}">

           </span>

         </div>

                        {{end}}

                        </div>
                {{else}}
                        <div id="currentlyNoIMAGESID">
                                <div class="centertext">
                                No uploads, you should start creating your first <a
href="/imgPool">pool</a>
                                <br>
                                and then create a <a href="/mosaic">Mosaic</a> :)
                                </div>
                                <div><img class="noImages" id="noImages"
src="Icons/cuteGolangs.png"></div>
                        </div>
                {{end}}
    </div>
    <div class="row footer">
      <p>Webprogrammierung Hausarbeit &copy; <b>Duy Khoi Nguyen</b></p>
    </div>
        </div>
        <!--modale-->
  <div id="imageModal" class="imageModal">
    <span class="close">&times;</span>
    <img class="imagemodal-content" id="imgModalID">
    <img class="deleteIMG" src="Icons/trash-can.png" id="deleteIMG">
  </div>
  <div id="imgInfoModalID" class="imgInfoModal">
    <span class="close">&times;</span>
    <div id="imgInfoText">Hallo</div>
  </div>
  <!--modale-->
        </body>
</html>`
var mosaicpage = `
```

```
<!DOCTYPE html>

<html>

        <head>

  <link rel="stylesheet" href="CSS_FONTS/picxStyle.css">

  <script src="JS/PICX.js"></script>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <meta http-equiv="X-UA-Compatible" content="ie=edge">

  <title>PICX Hausarbeit Webprogrammierung</title>

        </head>

        <body>

  <div class="box">

    <header class="row header">

      <div class="logo links">PICX</div>

      <nav class="center">

        <ul class="nav_center">

                                    <li id="galleryID"><a id="AGallery" href="/gallery">GALLERY</a></li>

                                    <li id="baseMotifsID"><a id="ABaseMotifs"
href="/baseMotive">BASIC MOTIVES</a></li>

                                    <li id="baseCreationID"><a id="AMosaic" href="/mosaic">MOSAIC
CREATION</a></li>

                                    <li id="imagePoolID"><a id="AImgPool" href="/imgPool">IMAGE
POOL</a></li>

        </ul>

      </nav>

      <div class="rechts directionColumn" id="rechts">

        <li><a class="logoutA" id="profile"><img id="logout" src="Icons/profile.png"> &#9660;
</a></li>

        <ul id="submenu1">

          <li><a id="settingsID" href="/settings"><img class="submenuIMAGE"
src="Icons/settings.png">setting</a></li>
```

```
            <li><a id="logoutID" href="/logout"><img class="submenuIMAGE"
src="Icons/logout.png">logout</a></li>

          </ul>

        </div>

      </header>

      <div class="row content">

        <div class="siteTitle"> MOSAIC CREATION</div>

                          <br>

                          {{if .PoolFeedback}} <div id="poolFeed" style="color:{{.FeedColor}};">
{{.PoolFeedback}} </div>

                          {{else}}

                          <!--<div id="notePOOL"> Note: You should have about 100 Images in one
pool to create a decent Mosaics. </div>-->

                          <div id="notePOOL"> Note: Default of "Use Kacheln" is Multiple Times and
optional. </div>

                          {{end}}

                  <form id="mosaicFormID" method="post" action="/mosaic"
enctype="multipart/form-data">

                          <fieldset id="mosaic-fieldset" >

                                  <legend>Upload to Mosaic</legend>


                                  <input type="file" name="mosaicfile" id="myfiles">

                                  <select name="selectedPool" id="selectedPoolID">

            <!--https://stackoverflow.com/questions/9447134/default-text-which-wont-be-shown-in-
drop-down-list-->

                                          <option selected disabled hidden>Choose Pool
here</option>

                                          {{range $i, $name := .PoolNames}}

                                          <option value="{{index $.ShowKachelSize
$i}}.{{$name}}">{{$name}} {{index $.PictureCount $i}}

                                          ({{index $.ShowKachelSize $i}}x{{index $.ShowKachelSize $i}})
</option>

            {{end}}
```

```
                </select>
                <select name="kachelmode" id="kachelmodeID" title="Option to use
Kacheln in Pool multiple times or just ones">
                    <option selected disabled hidden>Use Kacheln</option>
                    <option value="multiple times">Multiple Times</option>
                    <option value="one time">One Time</option>
                </select>

                <span id="albumMosaicSpan">
                <div class="dropdown">
                    <img class="dropbtn" id="createAlbumIMG"
src="Icons/plus.png"
                        title="Create a Album where to save Images" />
                    <div id="myDropdown" class="dropdown-content">
                        <div class="displayFlex">
                            <div id="newAlbumnameDIV">
                            <input type="text" placeholder="Enter name
of Album" id="newAlbumName"/>
                            </div>
                            <div id="createAlbumBtnDIV"> <span
id="creatAlbumBTN" type="button"
                                    value="create
Album">create Album</span></div>
                        </div>
                    </div>
                </div>
                <select name="chooseAlbum" id="chooseAlbumID" title="Choose a
Album where to save Images">
                    <option selected disabled hidden>Choose Album</option>
                    {{range $i, $album := .Albums}}
                    <option value="{{$album}}">{{$album}}</option>
                    {{end}}
                </select>

                </span>


                <input type="submit" id="upload_Btn" name="submitMosaic"
value="los geht's">

            </fieldset>

        </form>


        {{if .BeforeSource}}

        <div id="previewTitle"><br></div>

        {{else}}

        <div id="previewTitle">Preview:</div>

        {{end}}


        <div id="beforeAfterMosaicDiv">

            <div class="grid-Mosaic-Child">

                {{if .BeforeSource}} <img class="grid-img-MosaicC" id=""
src="{{.BeforeSource}}">

                {{else}}

                <div class="beforeAfterMosaic beforeAfterBorder"><span
class="unselectable">Before</span></div>

                {{end}}

            </div>

            <div class="grid-Mosaic-Child">

                <div class="beforeAfterMosaic"><span
class="unselectable">&#187;</span></div>

            </div>

            <div class="grid-Mosaic-Child">

                {{if .AfterSource}} <img class="grid-img-MosaicC" id=""
src="{{.AfterSource}}">

                {{else}}
```

```
                                        <div class="beforeAfterMosaic beforeAfterBorder"><span
class="unselectable">After</span></div>

                                        {{end}}

                              </div>

                    </div>

                              <div id="imageModal2" class="imageModal2">

                                        <span class="close">&times;</span>

                                        <img class="imagemodal-content" id="imgModalID" src="">

                              </div>


        <div id="loadermodal" class="loadermodal">

          <div class="loader" id="loaderModalID"></div>

        </div>

      </div>

   </div>

   </div>

          </body>

</html>`

var imgPoolpage = `

<!DOCTYPE html>

<html>

          <head>

   <link rel="stylesheet" href="CSS_FONTS/picxStyle.css">

   <script src="JS/PICX.js"></script>

   <meta charset="UTF-8">

   <meta name="viewport" content="width=device-width, initial-scale=1.0">

   <meta http-equiv="X-UA-Compatible" content="ie=edge">

   <title>PICX Hausarbeit Webprogrammierung</title>

          </head>

          <body>
```

```
<div class="box">
<header class="row header">
<div class="logo links">PICX</div>
<nav class="center">
    <ul class="nav_center">
    <li id="galleryID"><a id="AGallery" href="/gallery">GALLERY</a></li>
    <li id="baseMotifsID"><a id="ABaseMotifs" href="/baseMotive">BASIC MOTIVES</a></li>
    <li id="baseCreationID"><a id="AMosaic" href="/mosaic">MOSAIC CREATION</a></li>
    <li id="imagePoolID"><a id="AImgPool" href="/imgPool">IMAGE POOL</a></li>
    </ul>
</nav>
<div class="rechts directionColumn" id="rechts">
    <li><a class="logoutA" id="profile"><img id="logout" src="Icons/profile.png"> &#9660; </a></li>
    <ul id="submenu1">
        <li><a id="settingsID" href="/settings" ><img class="submenuIMAGE" src="Icons/settings.png">setting</a></li>
        <li><a id="logoutID" href="/logout"><img class="submenuIMAGE" src="Icons/logout.png">logout</a></li>
    </ul>
</div>
</header>
<div class="row content">
<div class="siteTitle"> IMAGE POOL</div>
<br>
{{if .PoolFeedback}} <div id="poolFeed" style="color:{{.FeedColor}};"> {{.PoolFeedback}} </div>
{{else}}
<div id="notePOOL"> Note: Don't forget to choose a Kachelsize. You can also generate random Pool Images with the Pool Generator. </div>
```

```
        {{end}}
        <form id="imgPoolFormID" method="post" action="/imgPool"  enctype="multipart/form-
data">
                <fieldset id="fieldsetImgPool">
                <legend>Upload to Image Pool</legend>
                <div id="fieldset-flex-Div">
                <input type="hidden" id="poolNameID" name="poolName" value="">
                <input type="file" name="myImgPoolfiles" id="myfiles" multiple="multiple">
                <select name="selectedKachelSize" id="kachelSizeImg-pool">
                <option selected disabled hidden>choose Kachelsize</option>
                {{range $i, $kachel := .Kachelsizes}}
                <option value="{{$kachel}}"> {{$kachel}} x {{$kachel}} </option>
                {{end}}
                </select>
                <input type="button" id="showPoolModulIDbtn" value="Pool/Upload">
                <input type="submit" id="uploadPool_Btn" name="submitPool" value="upload">
                <span class="flex-span-right">
                        <input type="button" id="showPoolModulGeneratorBtn" value="Pool
Generator" title="Generate random Kacheln in a Pool">
                </span>
                </div>
                </fieldset>
        </form>
        <div class="grid-containerImgPool" id="gridBoxImgPool">
        {{range $i, $name := .PoolNames}}


                <div class="grid-imgPools-item">
                        <span class="center-flex show-imgPools-DataA" id="{{index
$.ShowKachelSize $i}}.{{$name}}">
                                <a>{{$name}}   {{index $.PictureCount $i}} ({{index $.ShowKachelSize
$i}}x{{index $.ShowKachelSize $i}}) </a>
```

```
                              </span>

                              <div class="right-flex"><img id="Graph.{{index $.ShowKachelSize
$i}}.{{$name}}" class="barIMAGE" src="Icons/bar-graph.png"></div>

                    </div>


          {{end}}

          </div>


          <div id="poolModulID" class="poolModulClass">

                    <span class="close">&times;</span>

                    <div id="choosePool-modalContent">

                              <div id="poolModalTitle"> Choose Pool</div>

                              <div class="pool-scroll-Container">

                                        {{range $i, $name := .PoolNames}}

                                        <div class="poolChooseDiv">

                                                  <input class="poolChooseClass" type="radio" id="{{index
$.ShowKachelSize $i}}.{{$name}}" name="PoolRadio" value="{{$name}}">

                                                  <label for="{{$name}}"> {{$name}} {{index $.PictureCount
$i}} ({{index $.ShowKachelSize $i}}x{{index $.ShowKachelSize $i}})</label>

                                        </div>

                                        {{ end }}


                              </div>

                              <div id="poolModalcreate">

                                        <div id="plusCreatePool"><img id="addnewPoolID"
src="Icons/plus.png" alt=".">Create New Pool

                                        </div>

                                        <div id="createPoolbtnDiv">

                                                  <span class="addToPoolbtn" id="addToPoolbtn"
name="addToPoolbtn">

                                                            Finished

                                                  </span>
                    </div>
                </div>
                <div id="poolModalcreate2">
                    <div id="newPoolNameTitle">Pool Name</div>
                    <input class="createPoolname" type="text" id="createPoolname"
name="createPoolname">
                    <div id="createPoolbtnDiv">
                        <span class="createPoolbtn" id="createPoolbtn"
name="createPoolbtn" value="create/add Pool">
                            Create/Add Pool
                        </span>
                    </div>
                </div>
            </div>
        </div>

        <div id="poolModalshowData">
            <span class="close">&times;</span>
            <div id="poolModal-ContentData">
            <!-- Hier kommt das modal template-->
            </div>
        </div>

            <div id="poolGenerator-Modal">
                <span class="close">&times;</span>
       <form id="poolGenerator-Content-Modal" method="post"
action="/imgPool?getRandom=yes">
         <div id="poolGenerator-Title"> Pool Generator</div>
         <div id="generator-inputsDIV">
           <div class="generator-poolname">Poolname: </div>
           <div id="generator-select-input-Div">
```

```
            <input type="text" id="poolGenerator_name" name="poolname">
            <select name="kachelsize">
              <option selected disabled hidden>Size</option>
              <option value="5">5x5</option>
              <option value="10">10x10</option>
              <option value="15">15x15</option>
              <option value="20">20x20</option>
              <option value="25">25x25</option>
              <option value="30">30x30</option>
            </select>
          </div>
          <div class="generator-kachelnumb"> Kachel Number:</div>
          <div id="generator-size-Div">
            <input type="number" min=1 id="poolGenerator_KachelCount" name="kachelCount">
          </div>
        </div>
        <div id="generator_btnDiv">
          <div id="generator_submitDiv">
            <input type="submit" id="poolGenerator_Btn" name="generatePool" value="Generate
Pool">
          </div>
        </div>
      </form>
              </div>


              <div id="showGraph-Pool-Img-Modal">
              <span class="close">&times;</span>
                      <div id="graph-PoolModal-content">


                      </div>
```

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

var settingspage = `
<!DOCTYPE html>
<html>
            <head>
    <link rel="stylesheet" href="CSS_FONTS/picxStyle.css">
    <script src="JS/PICX.js"></script>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>PICX Hausarbeit Webprogrammierung</title>
            </head>
            <body>
    <div class="box">
        <header class="row header">
            <div class="logo links">PICX</div>
            <nav class="center">
                                    <ul class="nav_center">
                                        <li id="galleryID"><a id="AGallery"
href="/gallery">GALLERY</a></li>
                                        <li id="baseMotifsID"><a id="ABaseMotifs"
href="/baseMotive">BASIC MOTIVES</a></li>
                                        <li id="baseCreationID"><a id="AMosaic"
href="/mosaic">MOSAIC CREATION</a></li>
                                        <li id="imagePoolID"><a id="AImgPool"
href="/imgPool">IMAGE POOL</a></li>
```

```
                        </ul>

    </nav>

                    <div class="rechts directionColumn" id="rechts">

                        <li><a class="logoutA" id="profile"><img id="logout"
src="Icons/profile.png"> &#9660; </a></li>

                        <ul id="submenu1">

                            <li><a id="settingsID" href="/settings"><img
class="submenuIMAGE" src="Icons/settings.png">setting</a></li>

                            <li><a id="logoutID" href="/logout"><img
class="submenuIMAGE" src="Icons/logout.png">logout</a></li>

                        </ul>

                    </div>

  </header>

  <div class="row content">

    <div class="siteTitle settingstitle">Settings</div>

    <div class="centerContantDiv">

      <div class="flexRow">

        <nav class="sidenav">

          <ul>

                                        <li id="profileIDsettings">Profile</li>

            <li id="passwordsettings">Change Password</li>

            <li id="deleteACCsettings">Delete Account</li>

          </ul>

        </nav>

        <div id="settingsContent">

                                <div id="settingTitle" class="settingTitle">

            Hello {{.Username}}

            <img id="helloIcon" src="Icons/hello.png">

          </div>

          <div id="settingWelcomeText">

            In your settings, you can change Password.<br>
```

```
              Or delete your Account if you want to leave us :(
             </div>
            </div>
           </div>
          </div>
         </div>
         <div class="row footer">
          <p>Webprogrammierung Hausarbeit &copy; <b>Duy Khoi Nguyen</b></p>
         </div>
        </div>
           </body>
</html>
`

var changePWTemplate = `<div id="settingTitle" class="settingTitle">
           Change Password
           <img class="iconKeyDelete" src="Icons/key.png">
           </div>
           <div class="centerForm">
           <form id="changePasswordForm" class="cdForm" name="changePW">
                      <input type="password" name="oldPassword" placeholder="Old Password">
                      <input type="password" name="newPassword" placeholder="New Password">
                      <input type="password" name="newPassword2" placeholder="Verify New
Password">
                      <input type="button" id="changePWBtnID" value="change password">
           </form>
           {{if .Feedback}}<div class="feedbackstring"
style="color:{{.Color}};">{{.Feedback}}</div>{{end}}
</div>
`

var deleteAccTemplate = `<div id="settingTitle" class="settingTitle">
           Delete Account
        <img class="iconKeyDelete" src="Icons/deleteAcc.png">

        </div>

        <div class="centerForm">

        <form id="deleteAccForm" class="cdForm" name="deleteAcc">

                <input type="password" name="password" placeholder="Password">

                <input type="password" name="password2" placeholder="Verify Password">

                <input type="button" id="deleteAccBtnID" value="Delete Account">

        </form>

        {{if .Feedback}}<div class="feedbackstring"
style="color:{{.Color}};">{{.Feedback}}</div>{{end}}

</div>

`

var poolModalTemplate = `<div class="pool-modal-title">{{.Poolname}}</div>

                <div class="deletePoolDIV" id="deletePoolDIV"><img class="deleteWholePool"
src="Icons/trash-can.png" id="{{.CollectionName}}" alt="." title="delete Pool"></div>

                <div class="pooldata-scroll-Container">

                {{range $i, $img := .Images }}

                        <div class="pooldataDiv">

                                <img class="kachelPic" id="" src="{{$img.Source}}" alt="."
title="{{$img.Filename}}">

                                <span class="kachelname"

                                        title="{{$img.Filename}}">

                                        {{$img.Filename}}

                                </span>

                                <div class="left-PoolDiv">

                                        <img class="deletePoolIMG" id="{{$img.DbFileDir}}"
src="Icons/delete.png" alt="." title="delete">

                                        <a class=""
href="/downloadPoolImg?downloadPoolImage={{$img.DbFileDir}}" >

                                        <img class="downloadPoolIMG" id="{{$img.DbFileDir}}"
src="Icons/download.png" alt="." title="download">

                                        </a>
        <img class="infoPoolIMG" id="" src="Icons/information.png" alt="."

        title="Bildgröße: {{$img.Length}} Bytes&#10;UploadDate: {{$img.UploadDate}} &#10;Name:
{{$img.Filename}} &#10;Auflösung: {{$img.AuflosungX}} x {{$img.AuflosungY}} &#10;Helligkeit:
{{$img.Metadata.Brightness}} &#10;Farbverteilung {R G B}: {{$img.Metadata.MiddleColorVec}}">

                    </div>

                </div>

                {{end}}

            </div>

</div>
`

var graphModalTemplate = `

        <div class="graph-Pool-Title">Average RGB-Brightness: <br>{{.Poolname}}</div>

        <svg id="pool-Graph" width="400" height="270">

            <rect width="400" height="270" style="fill:rgb(255, 255, 255);stroke-linejoin: round;"
/>

            <g class="coordinateAxis">

                <polyline points="30,10 30,265 390,265" style="fill:none;stroke-
width:1;stroke:rgb(155, 155, 155)" />

                <text x="0" y="15">255</text>

                <text x="0" y="147">127</text>

                <text x="15" y="265">0</text>

            </g>

            <g class="redValueG">

                <!--points="65,265 140,265 140,redy 65,redy"-->

                <title>Red: {{.AvgRGB.X}}</title>

                <polygon class="redPoly" points="65,265 140,265 140,{{.AvgDrawCoord.X}}
65,{{.AvgDrawCoord.X}}" />

            </g>

            <g class="greenValueG">

                <title>green: {{.AvgRGB.Y}}</title>

                <polygon class="greenPoly" points="175,265 250,265
250,{{.AvgDrawCoord.Y}} 175,{{.AvgDrawCoord.Y}}" />
```

```
				</g>
				<g class="blueValueG">
					<title>blue: {{.AvgRGB.Z}}</title>
					<polygon title={{.AvgDrawCoord.Z}} class="bluePoly" points="285,265 
360,265 360,{{.AvgDrawCoord.Z}} 285,{{.AvgDrawCoord.Z}}" />
				</g>
			</svg>
			<div id="avgRGBflex-container">
				<span id="avgRGBText">
					<span class="colRect redColRect"></span> Average red value: 
{{.AvgRGB.X}}<br>
					<span class="colRect greenColRect"></span> Average green value: 
{{.AvgRGB.Y}}<br>
					<span class="colRect blueColRect"></span> Average blue value: 
{{.AvgRGB.Z}}<br>
					<span class="colRect"></span> Average brightness: {{.AvgBrightness}}<br>
				</span>
			</div>
`


var t = template.Must(template.ParseFiles("PICX.html")) //startseite
var dbName = "DB_Duy_Khoi_Nguyen_MatrikelNR_630305"
var server = "localhost" //in der HS "mongodb://borsti.inf.fh-flensburg.de:27017" verwenden
var userCredCol = "UserCredentials"
var dbNamePics = "DB_Duy_Khoi_Nguyen_MatrikelNR_630305_Pictures"
var poolFsName = "pool"
var mosaicFsName = "mosaic"
var baseImgFsName = "base"
var feedback = LoginSignInFeedback{}


//-------------------cookie namen--------------------------------------------------------------------------------
```

```
var currentUser = "CurrentUser"

var currentKachelSize = "currentKachelSize"

var currentPool = "currentMosaicPool"

var currentKachelMode = "currentKachelMode"

var currentChooseAlbum = "currentChooseAlbum"

var currentAlbum = "currentAlbum"


//-------------------variablen für lineare interpolation-für generateRandomRGB funktion---------------------
---------

var y2 = 16777215

var x2 = 126

var x1 = 32

var y1 = 0

var k = (y2 - y1) / (x2 - x1)


//Page for changePWSite---------------------------------------------------------------------------------------------------
----------------------------------

func changePWSite(w http.ResponseWriter, r *http.Request) {

	t := template.New("newPage")

	t, _ = t.Parse(changePWTemplate)

	t.Execute(w, feedback)

}


//Page for deleteAccSite---------------------------------------------------------------------------------------------------
---------------------------------

func deleteAccSite(w http.ResponseWriter, r *http.Request) {

	t := template.New("newPage")

	t, _ = t.Parse(deleteAccTemplate)

	t.Execute(w, feedback)

}
```

```
//Page for settingsPageHandler-----------------------------------------------------------------------------------------
------------------------------------------

func settingsPageHandler(w http.ResponseWriter, r *http.Request) {

	userCookie, err := r.Cookie(currentUser)

	if err != nil {

		return

	}

	dbSession, _ := mgo.Dial(server)

	defer dbSession.Close()

	// Datenbank wählen oder neu erstellen:

	db := dbSession.DB(dbName)

	collection := db.C(userCredCol)

	//check if Album already exists

	var user = UserCredential{}

	collection.FindId(bson.ObjectIdHex(userCookie.Value)).One(&user)

	sendUser := UserCredential{Username: user.Username}

	t := template.New("newPage")

	t, _ = t.Parse(settingspage)

	t.Execute(w, sendUser)

}


//Page for BaseMotifs---------------------------------------------------------------------------------------------------
------------------------------

func baseMotifPageHandler(w http.ResponseWriter, r *http.Request) {

	user, err := r.Cookie(currentUser)

	if err != nil {

		feedback.Feedback = ""

		t.ExecuteTemplate(w, "PICX.html", feedback)

		return

	}

	gridfsName := baseImgFsName + "." + user.Value
```

```
		files := retrieveImagesandReturnFileStrc(w, r, gridfsName, "BASIC MOTIVES", user.Value)
		t := template.New("newPage")
		t, _ = t.Parse(wholeGalleryPage)
		t.Execute(w, files)
}

//Page for Mosaic Image Page-----------------------------------------------------------------------------------------
--------------------------------

func galleryPageHandler(w http.ResponseWriter, r *http.Request) {
		keks, _ := r.Cookie(currentUser)
		gridfsName := mosaicFsName + "." + keks.Value
		files := retrieveImagesandReturnFileStrc(w, r, gridfsName, "GALLERY", keks.Value)
		t := template.New("newPage")
		t, _ = t.Parse(wholeGalleryPage)
		t.Execute(w, files)
}

//Page for Mosaic creation---------------------------------------------------------------------------------------------
-----------------------------

func mosaicPageHandler(w http.ResponseWriter, r *http.Request) {
		user, err := r.Cookie(currentUser)
		if err != nil {
				//startseite
				feedback.Feedback = ""
				t.ExecuteTemplate(w, "PICX.html", feedback)
				return
		}
		switch r.Method {
		case "GET":
				var poolnames = MosaicStrc{}
		poolnames.PoolNames, poolnames.PictureCount, poolnames.ShowKachelSize =
getpoolNames(w, r)
		poolnames.Albums = getUserAlbums(user.Value)
		t := template.New("mosaicPage")
		t, _ = t.Parse(mosaicpage)
		t.Execute(w, poolnames)
	case "POST": // Daten der form empfangen, files verarbeiten
		uploadMosaicHandler(w, r)
	default:
		w.WriteHeader(http.StatusMethodNotAllowed)
	}
}

//-----------------------------------------------------------------------------------------------------------------------------
----------------------
func imgPoolPageHandler(w http.ResponseWriter, r *http.Request) {
	_, err := r.Cookie(currentUser)
	if err != nil {
		feedback.Feedback = ""
		t.ExecuteTemplate(w, "PICX.html", feedback)
		return
	}
	switch r.Method {
	case "GET":
		var poolnames = PoolNamesStrc{}
		poolnames.Kachelsizes = kachelsizes
		poolnames.PoolNames, poolnames.PictureCount, poolnames.ShowKachelSize =
getpoolNames(w, r)
		t := template.New("newPageimg")
		t, _ = t.Parse(imgPoolpage)
		t.Execute(w, poolnames)
		case "POST": // Daten der multipart-form empfangen, files speichern
			getRand := r.URL.Query().Get("getRandom")
			if getRand == "yes" {
				randomPoolGenerator(w, r)
			} else {
				cutPoolImages(w, r)
			}
		default:
			w.WriteHeader(http.StatusMethodNotAllowed)
		}
}

//---------------------------------------------------------------------------------------------------------------------------------
----------------------
func selectAlbumAndShow(w http.ResponseWriter, r *http.Request) {
		keks, _ := r.Cookie(currentUser)
		album := r.URL.Query().Get("album")
		//fmt.Println(album)
		page := r.URL.Query().Get("page") //mosaic oder base
		setCookie(w, currentAlbum, album)
		session, err := mgo.Dial(server)
		check_ResponseToHTTP(err, w)
		defer session.Close()
		db := session.DB(dbNamePics)
		var gridfsName string
		var galleryORbase string
		if page == mosaicFsName {
			gridfsName = mosaicFsName + "." + keks.Value
			galleryORbase = "GALLERY"
		} else {
```

```
		gridfsName = baseImgFsName + "." + keks.Value
		galleryORbase = "BASIC MOTIVES"
	}
	collection := db.C(gridfsName + ".files")
	var result []fileTemplateStrc
	var files = ImagesStrc{}
	if album != "All Images" && album != "" {
		query := collection.Find(bson.M{"metadata.album": album}).Sort("-uploadDate")
		query.All(&result)
	} else {
		query := collection.Find(nil).Sort("-uploadDate") //query nach uplaoddate desc
ordnen
		query.All(&result)
	}
	files = helperRetrieveImageStruct(galleryORbase, gridfsName, keks.Value, result)
	t := template.New("newPage")
	t, _ = t.Parse(wholeGalleryPage)
	t.Execute(w, files)
}

//---------------------------------------------------------------------------------------------------------------------
----------------------
func deleteAlbum(w http.ResponseWriter, r *http.Request) {
	keks, _ := r.Cookie(currentUser)
	album := r.URL.Query().Get("album")
	currentchoosenAlb, err := r.Cookie(currentChooseAlbum)
	if err == nil && currentchoosenAlb.Value == album {
		deleteCookie(w, currentChooseAlbum)
	}
	page := r.URL.Query().Get("page") //mosaic oder base
	deleteCookie(w, currentAlbum)
```

```
gridfsName := mosaicFsName + "." + keks.Value
gridfsName2 := baseImgFsName + "." + keks.Value
var result []fileTemplateStrc
dbSession, _ := mgo.Dial(server)
db := dbSession.DB(dbNamePics) //db for images
db2 := dbSession.DB(dbName)    //db for usercrential, where the albums are being saved
collection := db2.C(userCredCol)
gridfs := db.GridFS(gridfsName)
gridfs2 := db.GridFS(gridfsName2)
//get every picture that is in our album
query := gridfs.Find(bson.M{"metadata.album": album})
query.All(&result)
//remove every picture that is in the album by Id
for _, element := range result {
        gridfs.RemoveId(element.ID)  //remove mosaic img
        gridfs2.RemoveId(element.ID) // remove Base img
}
match := bson.M{"_id": bson.ObjectIdHex(keks.Value)}
change := bson.M{"$pull": bson.M{"albums": album}} //remove album
collection.Update(match, change)
defer dbSession.Close()
var fsName string
var galleryORBase string
if page == "base" {
        fsName = baseImgFsName + "." + keks.Value
        galleryORBase = "BASIC MOTIVES"
} else {
        fsName = mosaicFsName + "." + keks.Value
        galleryORBase = "GALLERY"
}
```

```
	gridfs3 := db.GridFS(fsName)

	query2 := gridfs3.Find(nil).Sort("-uploadDate") //query nach uplaoddate desc ordnen

	var result2 []fileTemplateStrc

	query2.All(&result2)

	files := helperRetrieveImageStruct(galleryORBase, fsName, keks.Value, result2)

	t := template.New("newPage")

	t, _ = t.Parse(wholeGalleryPage)

	t.Execute(w, files)

}


//Helperfunction for retrieveImagesandReturnFileStrc to set up the Images struct--------------------------
--------------------------------------------------

func helperRetrieveImageStruct(GallerypageORBase string, gridfsName string, userHexId string,
result []fileTemplateStrc) ImagesStrc {

	var files = ImagesStrc{}

	files.Images = result

	//fmt.Printf("%d Bilder in der Collection\n", len(result))

	for i, element := range result {

		//element.Source = "/gridGetImage?dbName=" + dbNamePics + "&gridfsName=" +
poolFsName + "." + keks.Value + "." + "poolname&fileName=" + element.Filename

		files.Images[i].Source = "/gridGetImage?dbName=" + dbNamePics + "&gridfsName="
+ gridfsName + "&fileName=" + element.Filename + "&idName=" + element.ID.Hex()

		files.Images[i].DbFileDir = gridfsName + "." + element.ID.Hex() + "." +
element.Filename

		files.Images[i].ID = element.ID

		files.Images[i].Aufloesung = element.Metadata.Aufloesung

		files.Images[i].Metadata = element.Metadata

	}

	files.Albums = getUserAlbums(userHexId)

	files.PageSite = GallerypageORBase

	return files

}
```

```
//function so that Gallery and BaseMotifPage retrieve the Image data-----------------------------------------
---------------------------------------------

func retrieveImagesandReturnFileStrc(w http.ResponseWriter, r *http.Request, gridfsnm string,
GallerypageORBase string, userHexId string) ImagesStrc {

	albumCookie, errCookie := r.Cookie(currentAlbum)

	session, err := mgo.Dial(server)

	check_ResponseToHTTP(err, w)

	defer session.Close()

	db := session.DB(dbNamePics)

	gridfsName := gridfsnm

	collection := db.C(gridfsName + ".files")

	var result []fileTemplateStrc

	if errCookie == nil && albumCookie.Value != "All Images" {

		fmt.Println(albumCookie.Value)

		query := collection.Find(bson.M{"metadata.album": albumCookie.Value}).Sort("-
uploadDate")

		query.All(&result)

	} else {

		query := collection.Find(nil).Sort("-uploadDate") //query nach uplaoddate desc
ordnen

		query.All(&result)

	}

	return helperRetrieveImageStruct(GallerypageORBase, gridfsName, userHexId, result)

}


//------------------------------------------------------------------------------------------------------------------------
----------------------

func cutPoolImages(w http.ResponseWriter, r *http.Request) {

	cookie, _ := r.Cookie(currentUser)

	poolname := r.PostFormValue("poolName")

	kachelsize := r.PostFormValue("selectedKachelSize")
	//check if poolsize can be coverted to numb, who now if user changes Html to submit invalid
input
	kachelsizeInt, err := strconv.Atoi(kachelsize)
	if err != nil {
		runImgPoolPageWithMessage(w, r, "Please, choose a Kachelsize", "red")
		return
	}
	sizeCookie := http.Cookie{Name: currentKachelSize, Value: kachelsize}
	http.SetCookie(w, &sizeCookie)
	if poolname == "" {
		runImgPoolPageWithMessage(w, r, "Bitte einen Pool auswählen, oder erstellen",
"red")
		return
	}
	err = r.ParseMultipartForm(200000) // grab the multipart form
	check_ResponseToHTTP(err, w)
	formdata := r.MultipartForm		// ok, no problem so far, read the Form data
	files := formdata.File["myImgPoolfiles"] // grab the filenames
	if len(files) == 0 {
		runImgPoolPageWithMessage(w, r, "Upload/Pool erstellen fehlgeschlagen, es
wurden keine Images gesendet", "red")
		return
	}
	// DB-Verbindung:
	session, err := mgo.Dial(server)
	check_ResponseToHTTP(err, w)
	defer session.Close()
	db := session.DB(dbNamePics)
	//GridFs-collection erstellen/wählen:
	gridfsName := poolFsName + "." + cookie.Value + "." + kachelsize + "." + poolname
	gridfs := db.GridFS(gridfsName)
	for i, _ := range files {
		// upload-files öffnen:
		uplFile, err := files[i].Open()
		defer uplFile.Close()
		check_ResponseToHTTP(err, w)
		//decode file into a Image
		img, _, err := image.Decode(uplFile)
		if err != nil {
			runImgPoolPageWithMessage(w, r, "Bearbeitung abgebrochen, Upload
beinhaltet falschen Dateitypen", "red")
			return
		}
		var dstimg image.Image
		b := img.Bounds()
		switch {
		case b.Max.Y < kachelsizeInt:
			dstimg = imaging.Resize(img, 0, kachelsizeInt, imaging.Box)
			if dstimg.Bounds().Max.X < kachelsizeInt {
				dstimg = imaging.Resize(img, kachelsizeInt, 0, imaging.Box)
			}
		case b.Max.X < kachelsizeInt:
			dstimg = imaging.Resize(img, kachelsizeInt, 0, imaging.Box)
			if dstimg.Bounds().Max.Y < kachelsizeInt {
				dstimg = imaging.Resize(img, 0, kachelsizeInt, imaging.Box)
			}
		case b.Max.Y < b.Max.X:
			dstimg = imaging.Resize(img, 0, kachelsizeInt, imaging.Box)
		default:
			dstimg = imaging.Resize(img, kachelsizeInt, 0, imaging.Box)
		}
```

```
			// crop from center
			centercropimg := imaging.CropCenter(dstimg, kachelsizeInt, kachelsizeInt)
			// create buffer
			buff := new(bytes.Buffer) //use a byte slice as an io.Writer and turn strings/byte
slices into io.Readers.
			// encode/write image to buffer
			err = png.Encode(buff, centercropimg)
			check_ResponseToHTTP(err, w)
			// convert buffer to reader
			reader := bytes.NewReader(buff.Bytes())
			// grid-file mit diesem Namen erzeugen:
			gridFile, err := gridfs.Create(files[i].Filename)
			//um die Mittlere farbe zu speichern, bzw andere felder gibt es die SetMeta
			midColorVec, brightness := getAvgImageColorAndBrightness(0, kachelsizeInt, 0,
kachelsizeInt, kachelsizeInt, centercropimg)
			var metadata = Metadatas{MiddleColorVec: midColorVec, Brightness: brightness,
Kachelsize: kachelsize}
			gridFile.SetMeta(metadata)
			defer gridFile.Close()
			check_ResponseToHTTP(err, w)
			// in GridFSkopieren: Writer dst, Reader src
			//writer: shove data in writer, modify, save, compress, marshal it data
			//reader: read data from somewhere, and to something with it -> example put data
into a writer
			_, err = io.Copy(gridFile, reader)
			check_ResponseToHTTP(err, w)
			err = gridFile.Close()
			check_ResponseToHTTP(err, w)
		}
		runImgPoolPageWithMessage(w, r, "Upload nach "+poolname+" Erfolgreich.", "green")
}
```

```
//-----------------------------------------------------------------------------------------------------------------------
----------------------

func runImgPoolPageWithMessage(w http.ResponseWriter, r *http.Request, poolFeedback string,
feedColor string) {

	var poolnames = PoolNamesStrc{}

	if poolFeedback != "" {

		poolnames.PoolFeedback = poolFeedback

		poolnames.FeedColor = feedColor

	}

	poolnames.PoolNames, poolnames.PictureCount, poolnames.ShowKachelSize =
getpoolNames(w, r)

	poolnames.Kachelsizes = kachelsizes

	t := template.New("newPageimg")

	t, _ = t.Parse(imgPoolpage)

	t.Execute(w, poolnames)

}


//-----------------------------------------------------------------------------------------------------------------------
----------------------

func getpoolNames(w http.ResponseWriter, r *http.Request) ([]string, []string, []string) {

	// DB-Verbindung:

	session, err := mgo.Dial(server)

	check_ResponseToHTTP(err, w)

	defer session.Close()

	db := session.DB(dbNamePics)

	cookie, _ := r.Cookie(currentUser)

	var pools []string

	var picCount []string

	var kachelsize []string

	//var poolNameCount = []PoolNameAndCount{}
	collectionPoolNames, err := db.CollectionNames()
	check_ResponseToHTTP(err, w)
	for _, element := range collectionPoolNames {
		s := strings.Split(element, ".")
		if s[0] == poolFsName && s[1] == cookie.Value && s[len(s)-1] == "files" {
			//anzahl der files
			docCount, _ := db.C(element).Count()
			var poolname string
			for _, getname := range s[3 : len(s)-1] {
				poolname += getname + "."
			}
			poolname = strings.TrimSuffix(poolname, ".")
			pools = append(pools, poolname)
			picCount = append(picCount, " ("+strconv.Itoa(docCount)+")")
			kachelsize = append(kachelsize, s[2])
		}
	}
	return pools, picCount, kachelsize
}


//-----------------------------------------------------------------------------------------------------------------------------
-----------------------
func drawPoolGraph(w http.ResponseWriter, r *http.Request) {
	//query is Graph.poolsize.poolname
	keks, _ := r.Cookie(currentUser)
	fmt.Println(keks.Value)
	poolnamequery := r.URL.Query().Get("drawGraph")
	split := strings.Split(poolnamequery, ".")
	var fsName string
	for _, getname := range split[2:len(split)] {
		fsName += getname + "."
	}
	ksize := split[1]
	fsName = strings.TrimSuffix(fsName, ".")
	collectionName := poolFsName + "." + keks.Value + "." + ksize + "." + fsName
	session, err := mgo.Dial(server)
	check_ResponseToHTTP(err, w)
	defer session.Close()
	db := session.DB(dbNamePics)
	collection := db.C(collectionName + ".files")
	fmt.Println(collectionName)
	//fmt.Printf("%f\n", red)
	var graphvals = Graphstrct{}
	var pools = []fileTemplateStrc{}
	collection.Find(nil).All(&pools)
	var red float64
	var green float64
	var blue float64
	var brightness float64
	var teiler = 1
	for _, element := range pools {
		red += float64(element.Metadata.MiddleColorVec.X)
		green += float64(element.Metadata.MiddleColorVec.Y)
		blue += float64(element.Metadata.MiddleColorVec.Z)
		brightness += element.Metadata.Brightness
		teiler += 1
	}
	//https://yourbasic.org/golang/round-float-2-decimal-places/
	red = math.Round((red/float64(teiler))*100) / 100
	green = math.Round((green/float64(teiler))*100) / 100
```

```
        blue = math.Round((blue/float64(teiler))*100) / 100
        brightness = math.Round((brightness/float64(teiler))*100) / 100
        coordred := 265 - (red + 10)
        coordgreen := 265 - (green + 10)
        coordblue := 265 - (blue + 10)
        graphvals.Poolname = fsName
        graphvals.AvgRGB = Vector3Df{X: red, Y: green, Z: blue}
        graphvals.AvgDrawCoord = Vector3Df{X: coordred, Y: coordgreen, Z: coordblue}
        graphvals.AvgBrightness = brightness
        t := template.New("")
        t, _ = t.Parse(graphModalTemplate)
        t.Execute(w, graphvals)
}

//-----------------------------------------------------------------------------------------------------------------------------------
-----------------------

func showPoolCollection(w http.ResponseWriter, r *http.Request) {
        keks, _ := r.Cookie(currentUser)
        poolnamequery := r.URL.Query().Get("poolnameID")
        split := strings.Split(poolnamequery, ".")
        poolsize := split[0]
        var poolname string
        for _, getname := range split[1:len(split)] {
                poolname += getname + "."
        }
        poolname = strings.TrimSuffix(poolname, ".")
        // DB-Verbindung:
        collectionName := poolFsName + "." + keks.Value + "." + poolnamequery
        executePoolModalTemplate(w, r, collectionName, poolsize, poolname)
}
```

```
//wenn man eine Basismotiv lösche wird das dazugehöre Mosaic auch gelösche, vice versa--------------
-----------------------------------------------------------

func deleteBasicAndMosaicImage(w http.ResponseWriter, r *http.Request) {
	keks, _ := r.Cookie(currentUser)
	deletequery := r.URL.Query().Get("delete")
	//query ist z.B. base.5ddc211aa6022e0c693ed112.hexString.ImageName.jpg
	//query ist z.B. mosaic.5ddc211aa6022e0c693ed112.hexString.ImageName.jpg
	split := strings.Split(deletequery, ".")
	collectionbegin := split[0]
	hexstring := split[2]
	if keks.Value == split[1] {
		//db verbinden
		session, err := mgo.Dial(server)
		check_ResponseToHTTP(err, w)
		defer session.Close()
		db := session.DB(dbNamePics)
		//basismotiv löschen
		gridfs := db.GridFS(baseImgFsName + "." + split[1])
		err = gridfs.RemoveId(bson.ObjectIdHex(hexstring))
		check_ResponseToHTTP(err, w)
		//mosaicbild löschen
		gridfs2 := db.GridFS(mosaicFsName + "." + split[1])
		err = gridfs2.RemoveId(bson.ObjectIdHex(hexstring))
		check_ResponseToHTTP(err, w)
		switch collectionbegin {
		case baseImgFsName:
			gridfsName := baseImgFsName + "." + keks.Value
			files := retrieveImagesandReturnFileStrc(w, r, gridfsName, "BASIC MOTIVES",
keks.Value)

			t := template.New("newPage")
```

```
                    t, _ = t.Parse(wholeGalleryPage)

                    t.Execute(w, files)

                    //baseMotifPageHandler(w, r)

            case mosaicFsName:

                    galleryPageHandler(w, r)

            }

    }

}


//---------------------------------------------------------------------------------------------------------------------------
----------------------

func downloadBasicOrMosaicImage(w http.ResponseWriter, r *http.Request) {

    downloadquery := r.URL.Query().Get("download")

    //query ist z.B. base.5ddc211aa6022e0c693ed112.hexString.ImageName.jpg

    //query ist z.B. mosaic.5ddc211aa6022e0c693ed112.hexString.ImageName.jpg

    split := strings.Split(downloadquery, ".")

    collectionNm := split[0] + "." + split[1]

    hexString := split[2]

    var filename string

    for _, getfname := range split[3 : len(split)-1] {

            filename += getfname + "."

    }

    filename = strings.TrimSuffix(filename, ".")

    fileName := filename + "." + split[len(split)-1]

    downloadDateiHandler(w, r, collectionNm, hexString, fileName)

}


//---------------------------------------------------------------------------------------------------------------------------
----------------------

func deletePoolImageHandler(w http.ResponseWriter, r *http.Request) {

    poolImgdelete := r.URL.Query().Get("deletePoolImage")
```

```
	//query ist z.B.
pool.5ddc211aa6022e0c693ed112.30.Colorful.hexString.525d08554939731c3abf52e4fc01d1bc.jpg

	split := strings.Split(poolImgdelete, ".")

	var gridfsName = split[0] + "." + split[1] + "." + split[2] + "." + split[3]

	var hexstring = split[4]

	fmt.Println(gridfsName)

	// DB-Verbindung:

	session, err := mgo.Dial(server)

	check_ResponseToHTTP(err, w)

	defer session.Close()

	db := session.DB(dbNamePics)

	gridfs := db.GridFS(gridfsName)

	err = gridfs.RemoveId(bson.ObjectIdHex(hexstring))

	check_ResponseToHTTP(err, w)

	executePoolModalTemplate(w, r, gridfsName, split[2], split[3])

}


//--------------------------------------------------------------------------------------------------------------------------------
-----------------------

func deleteWholePoolHandler(w http.ResponseWriter, r *http.Request) {

	//query ist z.B pool.5de4cc0a76cd4c9a630d76c5.30.test

	poolName := r.URL.Query().Get("deletePool")

	split := strings.Split(poolName, ".")

	pool := split[2] + "." + split[3]

	fmt.Println(poolName)

	poolCookie, err := r.Cookie(currentPool)

	if err == nil && pool == poolCookie.Value {

		deleteCookie(w, currentPool)

	}

	session, err := mgo.Dial(server)

	check_ResponseToHTTP(err, w)
```

```
        defer session.Close()

        db := session.DB(dbNamePics)

        err = db.C(poolName + ".files").DropCollection()

        err = db.C(poolName + ".chunks").DropCollection()

        imgPoolPageHandler(w, r)

}


//-----------------------------------------------------------------------------------------------------------------------
-----------------------

func executePoolModalTemplate(w http.ResponseWriter, r *http.Request, collectionNm string,
kachelsize string, poolname string) {

        // DB-Verbindung:

        session, err := mgo.Dial(server)

        check_ResponseToHTTP(err, w)

        defer session.Close()

        db := session.DB(dbNamePics)

        collectionName := collectionNm

        collection := db.C(collectionName + ".files")

        var files = ImagesStrc{}

        var pools = []fileTemplateStrc{}

        collection.Find(nil).All(&pools)

        files.Images = pools

        for i, element := range pools {

                //element.Source = "/gridGetImage?dbName=" + dbNamePics + "&gridfsName=" +
poolFsName + "." + keks.Value + "." + "poolname&fileName=" + element.Filename

                files.Images[i].Source = "/gridGetImage?dbName=" + dbNamePics + "&gridfsName="
+ collectionName + "&fileName=" + element.Filename + "&idName=" + element.ID.Hex()

                files.Images[i].AuflosungX = kachelsize

                files.Images[i].AuflosungY = kachelsize

                files.Images[i].DbFileDir = collectionName + "." + element.ID.Hex() + "." +
element.Filename

                files.Images[i].IDHexstring = element.ID.Hex()
```

```
		}
		files.Poolname = poolname
		files.CollectionName = collectionName
		t := template.New("newPageimg")
		t, _ = t.Parse(poolModalTemplate)
		t.Execute(w, files)
}

//-----------------------------------------------------------------------------------------------------------------------------------
-----------------------
func uploadMosaicHandler(w http.ResponseWriter, r *http.Request) {
		cookie, _ := r.Cookie(currentUser)
		poolnamequery := r.PostFormValue("selectedPool")
		kachelmode := r.PostFormValue("kachelmode")
		album := r.PostFormValue("chooseAlbum")
		split := strings.Split(poolnamequery, ".")
		kachelsize, _ := strconv.Atoi(split[0])
		var poolname string
		for _, getname := range split[1:len(split)] {
				poolname += getname + "."
		}
		poolname = strings.TrimSuffix(poolname, ".")
		fmt.Println(poolname)
		fmt.Println(album)
		//poolnames.KachelSize = kachelsizes
		setCookie(w, currentPool, poolnamequery)
		setCookie(w, currentChooseAlbum, album)
		if poolname == "" {
				runMosaicPageWithMessage(w, r, "Upload Failed: Please choose a pool", "red")
				return
```

```
	}
	if album == "" {
		runMosaicPageWithMessage(w, r, "Upload Failed: Please choose a album", "red")
		return
	}
	// ParseMultipartForm parses a request body as multipart/form-data
	/*err := r.ParseMultipartForm(32 << 20)
	check_ResponseToHTTP(err, w)*/
	//file und headerinfo aus der form herauslesen
	file, header, err := r.FormFile("mosaicfile")
	if file == nil {
		runMosaicPageWithMessage(w, r, "Failed to Process: No Image has been sent",
"red")
		return
	}
	defer file.Close()
	if err == http.ErrMissingFile {
		runMosaicPageWithMessage(w, r, "Failed to Process: No Image has been sent",
"red")
		return
	}
	//file in Image decoden
	img, _, err := image.Decode(file)
	if err != nil {
		runMosaicPageWithMessage(w, r, "Failed to Process: Upload has wrong data
Extension", "red")
		return
	}
	//----------------------Base Motifs---------------------
	gridfsName := baseImgFsName + "." + cookie.Value
	preparedImg := prepareIMGforMosaic(img, kachelsize) //get prepared mosaic
	//----------------------------Mosaic---------------------------
	var mosaicImg image.Image
	if kachelmode == "one time" {
		mosaicImg, err = calculateCreateMosaic2(w, r, preparedImg, kachelsize) //get mosaic
		if err != nil {
			runMosaicPageWithMessage(w, r, err.Error(), "red")
			return
		}
	} else {
		mosaicImg = calculateCreateMosaic(w, r, preparedImg, kachelsize) //get mosaic
	}
	setCookie(w, currentKachelMode, kachelmode)
	gridID := saveIMGinDB(w, preparedImg, header.Filename, gridfsName, "", false, album) //id
vom ersten bild nehmen
	s1, _ := gridID.(bson.ObjectId)
	fmt.Println(s1.Hex())
	gridfsName2 := mosaicFsName + "." + cookie.Value
	gridID2 := saveIMGinDB(w, mosaicImg, header.Filename, gridfsName2, gridID, true, album)
//id des zweiten Bild mit den ersten gleichsetzen
	//Type assertions https://tour.golang.org/methods/15//https://yourbasic.org/golang/type-
assertion-switch/
	s, _ := gridID2.(bson.ObjectId)
	fmt.Println(s.Hex())
	var mosaicstrc = MosaicStrc{
		PoolFeedback: "Upload Successful",
		FeedColor:    "green",
		BeforeSource: "/gridGetImage?dbName=" + dbNamePics + "&gridfsName=" +
gridfsName + "&fileName=" + header.Filename + "&idName=" + s1.Hex(),
		AfterSource:  "/gridGetImage?dbName=" + dbNamePics + "&gridfsName=" +
gridfsName2 + "&fileName=" + header.Filename + "&idName=" + s.Hex(),
	}
	mosaicstrc.PoolNames, mosaicstrc.PictureCount, mosaicstrc.ShowKachelSize =
getpoolNames(w, r)

	mosaicstrc.Albums = getUserAlbums(cookie.Value)

	t := template.New("mosaicPage")

	t, _ = t.Parse(mosaicpage)

	t.Execute(w, mosaicstrc)

}


//--------------------------------------------------------------------------------------------------------------------------------
----------------------

func saveIMGinDB(w http.ResponseWriter, img image.Image, filename string, gridfsName string,
setID interface{}, setIdbool bool, album string) interface{} {

	session, err := mgo.Dial(server)

	check_ResponseToHTTP(err, w)

	defer session.Close()

	db := session.DB(dbNamePics)

	gridfs := db.GridFS(gridfsName)

	gridFile, err := gridfs.Create(filename) // grid-file mit diesem Namen erzeugen:

	if setIdbool {

		gridFile.SetId(setID)

	}

	check_ResponseToHTTP(err, w)

	buff := new(bytes.Buffer) //create buffer

	err = png.Encode(buff, img)

	var bound = img.Bounds()

	gridFile.SetMeta(bson.M{"aufloesung": strconv.Itoa(bound.Max.X) + "x" +
strconv.Itoa(bound.Max.Y), "album": album})

	check_ResponseToHTTP(err, w)

	reader := bytes.NewReader(buff.Bytes()) //convert buffer to reader

	_, err = io.Copy(gridFile, reader)	//Copy reader in GridFS

	check_ResponseToHTTP(err, w)
	buff.Reset() //reset Buffer

	gridFile.Close()

	return gridFile.Id()

}


//--------------------------------------------------------------------------------------------------------------------------------
-----------------------

func runMosaicPageWithMessage(w http.ResponseWriter, r *http.Request, poolFeedback string,
feedColor string) {

	cookie, _ := r.Cookie(currentUser)

	var poolnames = MosaicStrc{}

	if poolFeedback != "" {

		poolnames.PoolFeedback = poolFeedback

		poolnames.FeedColor = feedColor

	}

	poolnames.PoolNames, poolnames.PictureCount, poolnames.ShowKachelSize =
getpoolNames(w, r)

	poolnames.Albums = getUserAlbums(cookie.Value)

	t := template.New("mosaicPage")

	t, _ = t.Parse(mosaicpage)

	t.Execute(w, poolnames)

}


//--------------------------------------------------------------------------------------------------------------------------------
-----------------------

func prepareIMGforMosaic(img image.Image, kachelsize int) image.Image {

	//schneide die BasisMotive zurecht damit die 20x20 Kacheln auch draufpassen

	bounds := img.Bounds()

	restX := math.Mod(float64(bounds.Max.X), float64(kachelsize))

	newX := bounds.Max.X - int(restX)

	restY := math.Mod(float64(bounds.Max.Y), float64(kachelsize))
	newY := bounds.Max.Y - int(restY)

	centercropimg := imaging.CropCenter(img, newX, newY)

	return centercropimg

}


//Variante die Kacheln nur einmal benutzt------------------------------------------------------------------------------
--------------------------------------

func calculateCreateMosaic2(w http.ResponseWriter, r *http.Request, sourceImg image.Image,
kachelsize int) (image.Image, error) {

	fmt.Println(kachelsize)

	cookie, _ := r.Cookie(currentUser)

	poolname := r.PostFormValue("selectedPool")

	// DB-Verbindung:

	session, err := mgo.Dial(server)

	check_ResponseToHTTP(err, w)

	defer session.Close()

	db := session.DB(dbNamePics)

	//eines der Pool GridFs-collection wählen :

	gridfsName := poolFsName + "." + cookie.Value + "." + poolname

	gridfs := db.GridFS(gridfsName)

	var result = []fileTemplateStrc{}

	//alle Bilder aus dem Pool holen :

	gridfs.Find(nil).All(&result)

	check_ResponseToHTTP(err, w)

	bounds := sourceImg.Bounds()

	//check if there are enoughkacheln to use this mode

	var maxBx = bounds.Max.X

	var maxBy = bounds.Max.Y

	var neededkacheln = (maxBx / kachelsize) * (maxBy / kachelsize)

	fmt.Printf("neededded kacheln %d\n", neededkacheln)

	fmt.Printf("lenght of res %d\n", len(result))
	if neededkacheln >= len(result) {
		kstr := strconv.Itoa(kachelsize)
		return sourceImg, errors.New("Not enough Kacheln for Kachelmode: One Time; -
Kacheln needed: " + strconv.Itoa(neededkacheln) + " for size " + kstr + "x" + kstr)
	}
	rowNumb, colNumb := maxBx/kachelsize, maxBy/kachelsize
	var maxFarbLength = 10
	var farbabstaende = []Kachelstrct{}
	m := image.NewRGBA(image.Rect(0, 0, maxBx, maxBy))
	draw.Draw(m, m.Bounds(), sourceImg, image.Point{0, 0}, draw.Src)
	x2, y2 := kachelsize, kachelsize
	for y := 0; y < colNumb; y++ {
		for x := 0; x < rowNumb; x++ {
			farbVector, _ := getAvgImageColorAndBrightness(kachelsize*x, x2,
kachelsize*y, y2, kachelsize, sourceImg)
			for _, el := range result {
				var farbabstand = CalculateVectorDistance(farbVector,
el.Metadata.MiddleColorVec)
				if len(farbabstaende) < maxFarbLength {
					farbabstaende = append(farbabstaende, Kachelstrct{
						Farbabstand: farbabstand,
						ID:         el.ID,
					})
				} else {
					sort.Sort(FarbabstandSort(farbabstaende))
					if farbabstaende[maxFarbLength-1].Farbabstand >
farbabstand {
						farbabstaende[maxFarbLength-1].Farbabstand =
farbabstand
						farbabstaende[maxFarbLength-1].ID = el.ID
					}
				}
```

```
                        }
                        file, _ := gridfs.OpenId(farbabstaende[0].ID)
                        //remove object from result
                        removeIndex := findIndexofBSONid(farbabstaende[0].ID, result)
                        result = append(result[0:removeIndex], result[removeIndex+1:]...)
                        //file, _ := gridfs.OpenId(farbabstaende[0].ID)
                        defer file.Close()
                        kachel, _, _ := image.Decode(file)
                        farbabstaende = nil
                        draw.Draw(m, m.Bounds(), kachel, image.Point{-kachelsize * x, -kachelsize *
y}, draw.Over)
                        x2 += kachelsize
                }
                x2 = kachelsize
                y2 += kachelsize
        }
        return m, nil
}

//getElement index of slice-----------------------------------------------------------------------------------------------
--------------------------------
func findIndexofBSONid(element bson.ObjectId, data []fileTemplateStrc) int {
        for i, el := range data {
                if element == el.ID {
                        return i
                }
        }
        return -1 //not found.
}
```

```
//version mit Farbabstand-----------------------------------------------------------------------------------------------------
---------------------------------

func calculateCreateMosaic(w http.ResponseWriter, r *http.Request, sourceImg image.Image,
kachelsize int) image.Image {

        fmt.Println(kachelsize)

        cookie, _ := r.Cookie(currentUser)

        poolname := r.PostFormValue("selectedPool")

        // DB-Verbindung:

        session, err := mgo.Dial(server)

        check_ResponseToHTTP(err, w)

        defer session.Close()

        db := session.DB(dbNamePics)

        //eines der Pool GridFs-collection wählen :

        gridfsName := poolFsName + "." + cookie.Value + "." + poolname

        gridfs := db.GridFS(gridfsName)

        var result = []fileTemplateStrc{}

        //alle Bilder aus dem Pool holen (limit 1000 gesetzt):

        iter := gridfs.Find(nil).Limit(9000).Iter()

        err = iter.All(&result)

        check_ResponseToHTTP(err, w)

        iter.Close() //close Iter

        bounds := sourceImg.Bounds()

        rowNumb, colNumb := bounds.Max.X/kachelsize, bounds.Max.Y/kachelsize

        var maxFarbLength = 10

        var farbabstaende = []Kachelstrct{}

        m := image.NewRGBA(image.Rect(0, 0, bounds.Max.X, bounds.Max.Y))

        draw.Draw(m, m.Bounds(), sourceImg, image.Point{0, 0}, draw.Src)

        x2, y2 := kachelsize, kachelsize

        for y := 0; y < colNumb; y++ {

                for x := 0; x < rowNumb; x++ {
```

```
				farbVector, _ := getAvgImageColorAndBrightness(kachelsize*x, x2,
kachelsize*y, y2, kachelsize, sourceImg)
				for _, el := range result {
					var farbabstand = CalculateVectorDistance(farbVector,
el.Metadata.MiddleColorVec)
					if len(farbabstaende) < maxFarbLength {
						farbabstaende = append(farbabstaende, Kachelstrct{
							Farbabstand: farbabstand,
							ID:	el.ID,
						})
					} else {
						sort.Sort(FarbabstandSort(farbabstaende))
						if farbabstaende[maxFarbLength-1].Farbabstand >
farbabstand {
							farbabstaende[maxFarbLength-1].Farbabstand =
farbabstand
							farbabstaende[maxFarbLength-1].ID = el.ID
						}
					}
				}
				file, _ := gridfs.OpenId(farbabstaende[random(0, 6)].ID)
				//file, _ := gridfs.OpenId(farbabstaende[0].ID)
				defer file.Close()
				kachel, _, _ := image.Decode(file)
				farbabstaende = nil
				draw.Draw(m, m.Bounds(), kachel, image.Point{-kachelsize * x, -kachelsize *
y}, draw.Over)
				x2 += kachelsize
			}
			x2 = kachelsize
			y2 += kachelsize
		}
```

```
        return m
}

//-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------

func CalculateBrightness3DCol(n Vector3D) float64 { //auch vektorlänge genannt
        dx := float64(n.X)
        dy := float64(n.Y)
        dz := float64(n.Z)
        return math.Sqrt(dx*dx + dy*dy + dz*dz)
}

//-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------

func CalculateVectorDistance(n1 Vector3D, n2 Vector3D) float64 {
        //Farbabstand zwischen zwei Vektoren
        //vorsicht uint8 nimmt nur zahlen von 0 - 255
        dx := float64(n2.X) - float64(n1.X)
        dy := float64(n2.Y) - float64(n1.Y)
        dz := float64(n2.Z) - float64(n1.Z)
        return math.Sqrt(dx*dx + dy*dy + dz*dz)
}


//-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------

func random(min, max int) int {
        //http://golangcookbook.blogspot.com/2012/11/generate-random-number-in-given-
range.html
        //rand.Seed(time.Now().Unix())
        rand.Seed(time.Now().UTC().UnixNano())
```

```
	return rand.Intn(max-min) + min
}


//-----------------------------------------------------------------------------------------------------------------------------------------------------------


func (o BrightnessSort) Len() int          { return len(o) }
func (o BrightnessSort) Less(i, j int) bool { return o[i].Brightness < o[j].Brightness }
func (o BrightnessSort) Swap(i, j int)      { o[i], o[j] = o[j], o[i] }


func (o FarbabstandSort) Len() int          { return len(o) }
func (o FarbabstandSort) Less(i, j int) bool { return o[i].Farbabstand < o[j].Farbabstand }
func (o FarbabstandSort) Swap(i, j int)      { o[i], o[j] = o[j], o[i] }


//-----------------------------------------------------------------------------------------------------------------------------------------------------------

func getAvgImageColorAndBrightness(xstart int, xBound int, ystart int, yBound int, kachelsize int, i
image.Image) (Vector3D, float64) {
	var r, g, b uint32
	//bounds := i.Bounds()
	for y := ystart; y < yBound; y++ {
		for x := xstart; x < xBound; x++ {
			pr, pg, pb, _ := i.At(x, y).RGBA()
			r += pr //pixelrotanteil akkumulieren
			g += pg //pixelgelbanteil akkumulieren
			b += pb //pixelblauanteil akkumulieren
		}
	}
	d := uint32(kachelsize * kachelsize) //Kachelfläche
	r /= d
	g /= d
	b /= d
	var rgbVector = Vector3D{X: uint8(r / 0x101), Y: uint8(g / 0x101), Z: uint8(b / 0x101)}

	var brightness = CalculateBrightness3DCol(rgbVector)

	//color.NRGBA{uint8(r / 0x101), uint8(g / 0x101), uint8(b / 0x101), 255},

	return rgbVector, brightness

}




//-----------------------------------------------------------------------------------------------------------------------------------
-----------------------

func createAlbum(w http.ResponseWriter, r *http.Request) {

	//connect to user db with cookie hex

	album := r.URL.Query().Get("newAlbum")

	currentpool := r.URL.Query().Get("currentPool")

	currentmode := r.URL.Query().Get("currentMode")

	fmt.Println(currentpool)

	fmt.Println(currentmode)

	if album == "" || album == "All Images" {

		runMosaicPageWithMessage(w, r, "Failed to create Album, choose a valid name",
"red")

		return

	}

	//album könnte "Album 433   4343   5   " heißen, was anders wäre als "Album 433 4343 5"

	album = strings.Join(strings.Fields(album), " ")

	//album = strings.TrimRight(album, " ")

	cookie, _ := r.Cookie(currentUser)

	id := cookie.Value

	setCookie(w, currentPool, currentpool)

	setCookie(w, currentKachelMode, currentmode)

	// Verbindung zum Mongo-DBMS:

	dbSession, _ := mgo.Dial(server)
	defer dbSession.Close()

	// Datenbank wählen oder neu erstellen:

	db := dbSession.DB(dbName)

	collection := db.C(userCredCol)

	//check if Album already exists

	var albums UserCredential2

	collection.FindId(bson.ObjectIdHex(id)).One(&albums)

	fmt.Println(albums.Albums)

	for _, alb := range albums.Albums {

		if alb == album {

			runMosaicPageWithMessage(w, r, "failed to create Album, "+album+"
alreade exists", "red")

			return

		}

	}

	match := bson.M{"_id": bson.ObjectIdHex(id)}

	change := bson.M{"$push": bson.M{"albums": album}}

	setCookie(w, currentChooseAlbum, album)

	collection.Update(match, change)

	runMosaicPageWithMessage(w, r, "Success, created Album "+album, "green")

}


//-----------------------------------------------------------------------------------------------------------------------------
----------------------

func getUserAlbums(userHexId string) []string {

	dbSession, _ := mgo.Dial(server)

	defer dbSession.Close()

	// Datenbank wählen oder neu erstellen:

	db := dbSession.DB(dbName)

	collection := db.C(userCredCol)

	//check if Album already exists
	var albums UserCredential2

	collection.FindId(bson.ObjectIdHex(userHexId)).One(&albums)

	return albums.Albums

}


//------------------------------------------------------------------------------------------------------------------------------
-----------------------

func randomPoolGenerator(w http.ResponseWriter, r *http.Request) {

	cookie, _ := r.Cookie(currentUser)

	poolname := r.PostFormValue("poolname")

	kachelsize := r.PostFormValue("kachelsize")

	kachelcount := r.PostFormValue("kachelCount")

	kachelsizeInt, err := strconv.Atoi(kachelsize)

	if err != nil {

		runImgPoolPageWithMessage(w, r, "Generation failed, please choose a Kachelsize",
"red")

		return

	}

	kachelcountInt, err := strconv.Atoi(kachelcount)

	if err != nil {

		runImgPoolPageWithMessage(w, r, "Generation failed, please choose a valid
kachelnumber", "red")

		return

	}

	if poolname == "" {

		runImgPoolPageWithMessage(w, r, "Generation failed, please enter a Poolnamen",
"red")

		return

	}

	// DB-Verbindung:

	session, err := mgo.Dial(server)
```

```
check_ResponseToHTTP(err, w)
defer session.Close()
db := session.DB(dbNamePics)
gridfsName := poolFsName + "." + cookie.Value + "." + kachelsize + "." + poolname
gridfs := db.GridFS(gridfsName)
buff := new(bytes.Buffer)
// image generieren:
for x := 0; x < kachelcountInt; x++ {
        im := image.NewRGBA(image.Rect(0, 0, kachelsizeInt, kachelsizeInt))
        var r, g, b uint32
        // gesamtes image füllen:
        for x := 0; x < kachelsizeInt; x++ {
                for y := 0; y < kachelsizeInt; y++ {
                        randR, randG, randB := generateRandomRGB()
                        r += uint32(randR) //pixelrotanteil akkumulieren
                        g += uint32(randG) //pixelgelbanteil akkumulieren
                        b += uint32(randB) //pixelblauanteil akkumulieren
                        col := color.RGBA{randR, randG, randB, 255}
                        im.Set(x, y, col)
                }
        }
        d := uint32(kachelsizeInt * kachelsizeInt) //Kachelfläche
        r /= d
        g /= d
        b /= d
        //fmt.Printf("red:%d green:%d blue:%d\n", r, g, b)
        var rgbVector = Vector3D{X: uint8(r), Y: uint8(g), Z: uint8(b)}
        var brightness = CalculateBrightness3DCol(rgbVector)
        png.Encode(buff, im)            // encode/write image to buffer
        reader := bytes.NewReader(buff.Bytes()) // convert buffer to reader
```

```
				gridFile, err := gridfs.Create("randomKachel.jpg")
				var metadata = Metadatas{MiddleColorVec: rgbVector, Brightness: brightness,
Kachelsize: kachelsize, Aufloesung: kachelsize + "x" + kachelsize}
				gridFile.SetMeta(metadata)
				_, err = io.Copy(gridFile, reader)
				check_ResponseToHTTP(err, w)
				err = gridFile.Close()
				r, g, b = 0, 0, 0 //reset rgb
				buff.Reset()	//reset buffer
		}
		runImgPoolPageWithMessage(w, r, "Successfully created a random Kacheln", "green")
}

//---------------------------------------------------------------------------------------------------------------------------------
----------------------
func generateRandomRGB() (uint8, uint8, uint8) {
		linear_Interpolation := y1 + k*(random(32, 126)-x1) //interpolierter Wert
		meineUint32Farbe := uint32(linear_Interpolation)
		rot := uint8(meineUint32Farbe >> 16)
		green := uint8((meineUint32Farbe << 16) >> 24)
		blue := uint8((meineUint32Farbe << 16) >> 16)
		return rot, green, blue
}

func loginHandler(w http.ResponseWriter, r *http.Request) {
		loginErfolg := false
		name := r.PostFormValue("userLogName")
		password := r.PostFormValue("userLogPass")
		feedback := LoginSignInFeedback{}
		// Verbindung zum Mongo-DBMS:
		dbSession, _ := mgo.Dial(server)
```

```
        defer dbSession.Close()

        // Datenbank wählen oder neu erstellen:

        db := dbSession.DB(dbName)

        collection := db.C(userCredCol)

        var userCred []UserCredential2

        collection.Find(nil).All(&userCred)

        for _, user := range userCred {

                if user.Username == name && user.Password != password {

                        feedback.Feedback = "Kennwort falsch."

                        feedback.Color = "red"

                        t := template.New("feedbackTemplate")

                        t, _ = t.Parse(feedbackString)

                        t.Execute(w, feedback)

                        return

                }

                if user.Username == name && user.Password == password {

                        loginErfolg = true

                        //cookie setten

                        setCookie(w, currentUser, user.Id.Hex())

                        var files = ImagesStrc{}

                        gridfsName := mosaicFsName + "." + user.Id.Hex()

                        files = retrieveImagesandReturnFileStrc(w, r, gridfsName, "GALLERY",
user.Id.Hex())

                        t := template.New("feedbackTemplate")

                        t, _ = t.Parse(wholeGalleryPage)

                        t.Execute(w, files)

                        //weiterleitung zur gallery

                        return

                }

        }
```

```
	if !loginErfolg {
		feedback.Feedback = "User nicht registriert."
		feedback.Color = "red"
		t := template.New("feedbackTemplate")
		t, _ = t.Parse(feedbackString)
		t.Execute(w, feedback)
	}

}

func registerHandler(w http.ResponseWriter, r *http.Request) {
	userExistNot := true
	name := r.PostFormValue("userRegName")
	password := r.PostFormValue("userRegPass")
	// Verbindung zum Mongo-DBMS:
	dbSession, _ := mgo.Dial(server)
	defer dbSession.Close()
	// Datenbank wählen oder neu erstellen:
	db := dbSession.DB(dbName)
	collection := db.C(userCredCol)
	//define feedback
	feedback := LoginSignInFeedback{}
	// Userdaten aus der Datenbank auslesen
	var userCred []UserCredential
	collection.Find(nil).All(&userCred)
	if len(password) < 3 {
		feedback.Color = "red"
		feedback.Feedback = "Kennwort < 3 Zeichen"
		userExistNot = false
	} else {
```

```
            for _, user := range userCred {
                    if user.Username == name {
                            feedback.Color = "red"
                            feedback.Feedback = "User " + name + " existiert bereits."
                            userExistNot = false
                            break
                    }
            }
        }
        if userExistNot {
            doc := UserCredential{Username: name, Password: password}
            _ = collection.Insert(doc)
            feedback.Color = "green"
            feedback.Feedback = "User " + name + " registriert."
        }
        t := template.New("feedbackTemplate")
        t, _ = t.Parse(feedbackString)
        t.Execute(w, feedback)
}

func startHandler(w http.ResponseWriter, r *http.Request) {
        keks, err := r.Cookie(currentUser)
        if err != nil {
            feedback.Feedback = ""
            t.ExecuteTemplate(w, "PICX.html", feedback)
            return
        }
        dbSession, _ := mgo.Dial(server)
        defer dbSession.Close()
        // Datenbank wählen oder neu erstellen:
```

```
	db := dbSession.DB(dbName)
	collection := db.C(userCredCol)
	// Userdaten aus der Datenbank auslesen
	var userCred []UserCredential2
	collection.Find(nil).All(&userCred)
	//check if currentusercooke hex matches one hex of database
	for _, user := range userCred {
		if keks.Value == user.Id.Hex() {
			galleryPageHandler(w, r) //weiterleitung zur gallery
			break
		}
	}
}

func logoutHandler(w http.ResponseWriter, r *http.Request) {
	deleteCookie(w, currentUser)
	deleteCookie(w, currentKachelMode)
	deleteCookie(w, currentKachelSize)
	deleteCookie(w, currentPool)
	deleteCookie(w, currentChooseAlbum)
	deleteCookie(w, currentAlbum)
	fmt.Println("cookies deleted!")
	feedback.Feedback = ""
	t.ExecuteTemplate(w, "PICX.html", feedback)
}

func deleteCookie(w http.ResponseWriter, name string) {
	// Setting MaxAge<0 means delete cookie now.
	c := http.Cookie{
		Name:   name,
		MaxAge: -1}
	http.SetCookie(w, &c)
}

func setCookie(w http.ResponseWriter, name string, value string) {
	if value != "" {
		newCookie := http.Cookie{Name: name, Value: value}
		http.SetCookie(w, &newCookie)
	}
}

func downloadPoolImg(w http.ResponseWriter, r *http.Request) {
	poolNameAndImg := r.URL.Query().Get("downloadPoolImage")
	//query ist z.B.
pool.5ddc211aa6022e0c693ed112.30.Colorful.Hexstring.525d08554939731c3abf52e4fc01d1bc.jpg
	split := strings.Split(poolNameAndImg, ".")
	var gridfsName = split[0] + "." + split[1] + "." + split[2] + "." + split[3]
	fmt.Println(gridfsName)
	var hexString = split[4]
	var filename string
	for _, getfname := range split[5 : len(split)-1] {
		filename += getfname
	}
	fileName := filename + "." + split[len(split)-1]
	// DB-Verbindung:
	downloadDateiHandler(w, r, gridfsName, hexString, fileName)
}

func downloadDateiHandler(w http.ResponseWriter, r *http.Request, gridfsName string, hexstring
string, fileName string) {
	session, err := mgo.Dial(server)
	check_ResponseToHTTP(err, w)
	defer session.Close()
	db := session.DB(dbNamePics)
	//https://stackoverflow.com/questions/49118889/how-to-download-a-file-from-mongo-
gridfs-using-golang
	file, err := db.GridFS(gridfsName).OpenId(bson.ObjectIdHex(hexstring))
	check_ResponseToHTTP(err, w)
	fileSize := file.Size()
	dateiExt := ""
	contentType := ""
	if strings.Contains(fileName, ".") {
		split2 := strings.Split(fileName, ".")
		dateiExt = split2[len(split2)-1]
	} else {
		dateiExt = "unbekannt"
	}
	dateiExt = strings.ToLower(dateiExt)
	switch dateiExt {
	case "jpg":
		contentType = "image/jpeg"
	case "jpeg":
		contentType = "image/jpeg"
	case "png":
		contentType = "image/png"
	default:
		contentType = "application/octet-stream"
	}
	// Mit dem Content-Disposition header wird dem Browser mitgeteilt, die
	// folgende Datei nicht anzuzeigen, sondern in den download-Ordner zu kopieren:
	w.Header().Set("Content-Disposition", "attachment; filename="+fileName)
	w.Header().Set("Content-Type", contentType)
	w.Header().Set("Content-Length", strconv.FormatInt(fileSize, 10))
	// file in den ResponseWriter kopieren:
	io.Copy(w, file)
	err = file.Close()
	check_ResponseToHTTP(err, w)
}

//behandelt image-request an GridFS:-------------------------------------------------------------------------------------------------------------------------

func getImageHandler(w http.ResponseWriter, r *http.Request) {
	// request lesen:
	r_dbName := r.URL.Query().Get("dbName")
	r_gridfsName := r.URL.Query().Get("gridfsName")
	r_fileName := r.URL.Query().Get("fileName")
	r_idName := r.URL.Query().Get("idName")
	// DB-Verbindung:
	session, err := mgo.Dial(server)
	check_ResponseToHTTP(err, w)
	defer session.Close()
	db := session.DB(r_dbName)
	// angeforderte GridFs-collection dieser DB:
	gridfs := db.GridFS(r_gridfsName)
	// file aus GridFS lesen und als response senden:
	//gridFile, err := gridfs.Open(r_fileName)
	gridFile, err := gridfs.OpenId(bson.ObjectIdHex(r_idName))
	check_ResponseToHTTP(err, w)
	// content-type header senden:
	tmpSlice := strings.Split(r_fileName, ".")
	fileExtension := tmpSlice[len(tmpSlice)-1] // das letzte Element
	fileExtension = strings.ToLower(fileExtension)
	var mimeType string
	switch fileExtension {
	case "jpeg", "jpg":
		mimeType = "image/jpeg"
	case "png":
		mimeType = "image/png"
	case "gif":
		mimeType = "image/gif"
	default:
		mimeType = "text/html"
	}
	w.Header().Add("Content-Type", mimeType)
	// image senden:
	_, err = io.Copy(w, gridFile)
	check_ResponseToHTTP(err, w)
	err = gridFile.Close()
	check_ResponseToHTTP(err, w)
}

//-------------------------------------------------------------------------------------------------------------------------------
-----------------------

func check_ResponseToHTTP(err error, w http.ResponseWriter) {
	if err != nil {
		fmt.Fprintln(w, err)
		http.Error(w, err.Error(), http.StatusInternalServerError)
		return
	}
}
```

```
//----------------------------------------------------------------------------------------------------------------------------------------------------
func changePassword(w http.ResponseWriter, r *http.Request) {
	keks, err := r.Cookie(currentUser)
	if err != nil {
		return
	}
	password := r.PostFormValue("oldPassword")
	newPassword1 := r.PostFormValue("newPassword")
	newPassword2 := r.PostFormValue("newPassword2")
	feedback := LoginSignInFeedback{}
	if newPassword1 != newPassword2 {
		feedback.Color = "red"
		feedback.Feedback = "New Passwords arent equal"
		t := template.New("newPage")
		t, _ = t.Parse(changePWTemplate)
		t.Execute(w, feedback)
		return
	}
	if len(newPassword1) < 3 {
		feedback.Color = "red"
		feedback.Feedback = "New Password is too short < 3 charackter"
		t := template.New("newPage")
		t, _ = t.Parse(changePWTemplate)
		t.Execute(w, feedback)
		return
	}
	dbSession, _ := mgo.Dial(server)
	defer dbSession.Close()
	db := dbSession.DB(dbName)
	collection := db.C(userCredCol)

	var userCred []UserCredential2

	collection.Find(nil).All(&userCred)

	//check if currentusercookie hex matches one hex of database

	for _, user := range userCred {

		if keks.Value == user.Id.Hex() && user.Password == password {

			collection.Update(bson.M{"_id": bson.ObjectIdHex(keks.Value)},
bson.M{"$set": bson.M{"password": newPassword1}})

			feedback.Color = "green"

			feedback.Feedback = "Password changed Succesfully"

			t := template.New("newPage")

			t, _ = t.Parse(changePWTemplate)

			t.Execute(w, feedback)

			return

		}

	}

	feedback.Color = "red"

	feedback.Feedback = "Entered Password was wrong"

	t := template.New("newPage")

	t, _ = t.Parse(changePWTemplate)

	t.Execute(w, feedback)

}


//-----------------------------------------------------------------------------------------------------------------------------
----------------------

func deleteAccount(w http.ResponseWriter, r *http.Request) {

	keks, err := r.Cookie(currentUser)

	if err != nil {

		return

	}

	password1 := r.PostFormValue("password")
```

```
password2 := r.PostFormValue("password2")
feedback := LoginSignInFeedback{}
if password1 != password2 {
        feedback.Color = "red"
        feedback.Feedback = "Passwords are unequal"
        t := template.New("newPage")
        t, _ = t.Parse(deleteAccTemplate)
        t.Execute(w, feedback)
        return
}
dbSession, _ := mgo.Dial(server)
defer dbSession.Close()
db := dbSession.DB(dbName)
collection := db.C(userCredCol)
var userCred []UserCredential2
collection.Find(nil).All(&userCred)
//check if currentusercookie hex matches one hex of database
for _, user := range userCred {
        if keks.Value == user.Id.Hex() && user.Password == password1 {
                db2 := dbSession.DB(dbNamePics)
                //alle namen aller Collection herauslesen
                collectionPoolNames, err := db2.CollectionNames()
                if err != nil {
                        break
                }
                //alle pools, basismotive und mosaik des nutzers löschen
                for _, element := range collectionPoolNames {
                        s := strings.Split(element, ".")
                        if s[1] == keks.Value {
                                db2.C(element).DropCollection()
                        }
                    }
                    //User aus der Collection entferen
                    collection.Remove(bson.M{"_id": bson.ObjectIdHex(keks.Value)})
                    logoutHandler(w, r)
                    return
                }
        }
        feedback.Color = "red"
        feedback.Feedback = "Entered Passwords were wrong"
        t := template.New("newPage")
        t, _ = t.Parse(deleteAccTemplate)
        t.Execute(w, feedback)
}

//-----------------------------------------------------------------------------------------------------------------------------------
-----------------------

func main() {
        //static Fileserver
        http.Handle("/", http.FileServer(http.Dir("static")))
        http.HandleFunc("/deleteAccount", deleteAccount)
        http.HandleFunc("/changePassword", changePassword)
        http.HandleFunc("/drawPoolGraph", drawPoolGraph)
        http.HandleFunc("/deleteAccSite", deleteAccSite)
        http.HandleFunc("/changePWSite", changePWSite)
        http.HandleFunc("/deleteAlbum", deleteAlbum)
        http.HandleFunc("/selectAlbumAndShow", selectAlbumAndShow)
        http.HandleFunc("/createAlbum", createAlbum)                    //
http://localhost:4242/createAlbum
        http.HandleFunc("/downloadMosaicOrBasic", downloadBasicOrMosaicImage) //
http://localhost:4242/downloadMosaicOrBasic
```

```
	http.HandleFunc("/deleteMosaicAndBasic", deleteBasicAndMosaicImage)   //
http://localhost:4242/deleteMosaicAndBasic

	http.HandleFunc("/gallery", galleryPageHandler)

	http.HandleFunc("/deleteWholePool", deleteWholePoolHandler) //
http://localhost:4242/deleteWholePool

	http.HandleFunc("/downloadPoolImg", downloadPoolImg)       //
	http://localhost:4242/downloadPoolImg

	http.HandleFunc("/deletePoolImg", deletePoolImageHandler)   //
http://localhost:4242/deletePoolImg

	http.HandleFunc("/gridGetImage", getImageHandler)          //
http://localhost:4242/gridGetImage

	http.HandleFunc("/showPool", showPoolCollection)

	http.HandleFunc("/settings", settingsPageHandler)

	http.HandleFunc("/baseMotive", baseMotifPageHandler)

	http.HandleFunc("/mosaic", mosaicPageHandler)

	http.HandleFunc("/imgPool", imgPoolPageHandler) //http://localhost:4242/imgPool

	http.HandleFunc("/logout", logoutHandler)

	http.HandleFunc("/login", loginHandler)

	http.HandleFunc("/register", registerHandler) // http://localhost:4242/register

	http.HandleFunc("/picx", startHandler)       // http://localhost:4242/picx

	err := http.ListenAndServe(":4242", nil)

	if err != nil {

		fmt.Println(err)

	}

}
```

2. Quellcode JS

```
window.addEventListener("load", function () {
   var xhr5 = new XMLHttpRequest();
   var xhr4 = new XMLHttpRequest();
   var xhr3 = new XMLHttpRequest();
   var xhr2 = new XMLHttpRequest();
   var xhr = new XMLHttpRequest();
   xhr.addEventListener("load", function () {
      if (xhr.responseText.substring(2, 22) === '<div id="feedbackID"') {
         document.getElementById('feedDIV').innerHTML = xhr.responseText;
      } else {
         document.getElementsByTagName("body")[0].innerHTML = xhr.responseText;
         loadImagePoolPageHandlers();
         loadscndPageListeners();
         loadfirstPageListeners();
         loadMosaicPageHandlers();
         loadSettingPageHandlers();
      }
      checkCookie()
   });
   xhr5.addEventListener("load", function () {
      if (xhr5.responseText.substring(0, 22) == '<div id="settingTitle"') {
         document.getElementById('settingsContent').innerHTML = xhr5.responseText;
         loadChangeandDeleteHandler();
      } else {
         //this happens when Acc is sucessfully deleted
         document.getElementsByTagName("body")[0].innerHTML = xhr5.responseText;
         loadfirstPageListeners();
      }
   });
   xhr4.addEventListener("load", function () {
      document.getElementById('poolModal-ContentData').innerHTML = xhr4.responseText;
      loadImagePoolPageHandlers();
      deleteImgfromPool();
   });
   xhr2.addEventListener("load", function () {
      document.getElementById('graph-PoolModal-content').innerHTML = xhr2.responseText;
      loadImagePoolPageHandlers();
   });
   xhr3.addEventListener("load", function () {
      console.log(xhr3.responseText);
      document.getElementById('feedDIV').innerHTML = xhr3.responseText;

   });

   var popUP = document.getElementById("popUPID");
   var popupContent = document.getElementById("pupup-contentID");


   function getCookie(name) {
        var nameEQ = name + "=";
        var ca = document.cookie.split(';');
        for (var i = 0; i < ca.length; i++) {
            var c = ca[i];
            while (c.charAt(0) == ' ') c = c.substring(1, c.length);
            if (c.indexOf(nameEQ) == 0) return c.substring(nameEQ.length, c.length);
        }
        return null;
    }

    checkCookie()
    function checkCookie() {
        var currentKachelsize = getCookie("currentKachelSize");
        var currentKachelmode = getCookie("currentKachelMode");
        var currentMosaicPool = getCookie("currentMosaicPool");
        var currentChoosenAlbum = getCookie("currentChooseAlbum");
        var curentAlbum = getCookie("currentAlbum");
        if (document.body.contains(document.getElementById("fieldsetImgPool")) && currentKachelsize
!= null) {
            document.getElementById("kachelSizeImg-pool").value = currentKachelsize;
        }
        if (document.body.contains(document.getElementById("mosaic-fieldset"))) {
            if (currentKachelmode != "" && currentKachelmode != null) {
                document.getElementById("kachelmodeID").value = currentKachelmode.slice(1, -1);
            }
            if (currentMosaicPool != null) {
                document.getElementById("selectedPoolID").value = currentMosaicPool;
            }
            if (currentChoosenAlbum != null) {
                if (currentChoosenAlbum.slice(-1) == '"' && currentChoosenAlbum.charAt(0) == '"') {
                    document.getElementById("chooseAlbumID").value = currentChoosenAlbum.slice(1, -1);
                } else {
                    document.getElementById("chooseAlbumID").value = currentChoosenAlbum
                }
            }
        }
        if (document.body.contains(document.getElementById("selectAlbumDiv"))) {
            var currentAlbum;
            if (curentAlbum != null) {
                if (curentAlbum.slice(-1) == '"' && curentAlbum.charAt(0) == '"') {
                    currentAlbum = curentAlbum.slice(1, -1);
                } else {
                    currentAlbum = curentAlbum
                }
                document.getElementById("albumSelection").value = currentAlbum

            }
            if (currentAlbum == null || currentAlbum == "All Images") {
                document.getElementById("deleteDropdownID").style.display = "none"
            }
        }

    }

    function deleteImgfromPool() {
        if (document.body.contains(document.getElementsByClassName("kachelPic")[0])) {
            document.querySelectorAll(".deletePoolIMG").forEach(function (poolImg) {
                poolImg.onclick = function () {
                    console.log(this.id);
                    xhr4.open("GET", "http://localhost:4242/deletePoolImg?deletePoolImage=" + this.id);
                    xhr4.send();
                }
            });
        }
    }

    loadfirstPageListeners();
    function loadfirstPageListeners() {
        if (document.body.contains(document.getElementById("reg"))) {
            popUP = document.getElementById("popUPID");
            popupContent = document.getElementById("pupup-contentID")
            document.getElementsByClassName("close")[0].addEventListener("click", function () {
                popUP.style.display = "none";
            });
            document.getElementById("reg").addEventListener("click", function () {
                createRegister();
                console.log("reg")
                popUP.style.display = "block";
            });
            document.getElementById("log").addEventListener("click", function () {
                createLogin();
                console.log("log")
                popUP.style.display = "block";
            });

        }
    }

    loadscndPageListeners();
    function loadscndPageListeners() {
        if (document.body.contains(document.getElementById("settingsID"))) {
            document.getElementById("settingsID").addEventListener("click", function () {
                console.log("settings")
            });
            giveIMGeventHandler();
        }

    }

    loadImagePoolPageHandlers();
```

```
 function loadImagePoolPageHandlers() {
    if (document.body.contains(document.getElementById('createPoolbtn'))) {

        var poolGen_btn = document.getElementById("showPoolModulGeneratorBtn");
        var poolGen_modal = document.getElementById("poolGenerator-Modal");
        poolGen_btn.addEventListener("click", function () {
            poolGen_modal.style.display = "block"
        });

        var poolModaldata = document.getElementById('poolModalshowData');
        var createPool = document.getElementById('createPoolbtn');
        var plusPool = document.getElementById('plusCreatePool');
        var addToPool = document.getElementById('addToPoolbtn');
        var showPoolModalBtn = document.getElementById('showPoolModulIDbtn');
        var poolModal = document.getElementById('poolModulID');
        var choosenPoolName = document.getElementById("poolNameID");
        plusPool.addEventListener("click", function () {
            document.getElementById('poolModalcreate').style.display = "none";
            document.getElementById('poolModalcreate2').style.display = "block";
        });
        createPool.addEventListener("click", function () {
            choosenPoolName.value = document.getElementById("createPoolname").value;
            document.getElementById('uploadPool_Btn').click();
        })
        addToPool.addEventListener("click", function () {
            var pools = document.getElementsByName('PoolRadio');
            var poolsVal;
            for (var i = 0; i < pools.length; i++) {
                if (pools[i].checked) {
                    poolsVal = pools[i].value;
                    choosenPoolName.value = poolsVal;
                    break;
                }
            }
            document.getElementById('uploadPool_Btn').click();
        })

        showPoolModalBtn.addEventListener("click", function () {
            poolModal.style.display = "block";
        })

        document.querySelectorAll(".show-imgPools-DataA").forEach(function (pool) {
            pool.onclick = function () {
                //onclick statt addEventListner, weil addEventlistener bei jedem click eine funktion
hinzufügt
                console.log(this.id);
                poolModaldata.style.display = "block";
                xhr4.open("GET", "http://localhost:4242/showPool?poolnameID=" + this.id);
                xhr4.send();
            }
```

```
});

var graphdata = document.getElementById("showGraph-Pool-Img-Modal")
document.querySelectorAll(".barIMAGE").forEach(function (graph) {
  graph.onclick = function () {
    console.log(graph.id);
    graphdata.style.display = "block";
    xhr2.open("GET", "http://localhost:4242/drawPoolGraph?drawGraph=" + this.id);
    xhr2.send();
  };
});

document.getElementsByClassName("close")[0].onclick = function () {
  poolModal.style.display = "none";
}
document.getElementsByClassName("close")[1].onclick = function () {
  poolModaldata.style.display = "none";
  document.getElementById('poolModal-ContentData').innerHTML = "";
}
document.getElementsByClassName("close")[2].onclick = function () {
  poolGen_modal.style.display = "none"
}
document.getElementsByClassName("close")[3].onclick = function () {
  graphdata.style.display = "none"
}

if (document.body.contains(document.getElementsByClassName("deleteWholePool")[0])) {
  document.getElementsByClassName("deleteWholePool")[0].onclick = function () {
    xhr.open("GET", "http://localhost:4242/deleteWholePool?deletePool=" + this.id);
    xhr.send();
  }
}

window.addEventListener("click", function (event) {
  switch (event.target) {
    case poolModal:
      document.getElementById('poolModalcreate').style.display = "block";
      document.getElementById('poolModalcreate2').style.display = "none";
      poolModal.style.display = "none";
      break
    case poolModaldata:
      poolModaldata.style.display = "none";
      document.getElementById('poolModal-ContentData').innerHTML = "";
      break
    case poolGen_modal:
      poolGen_modal.style.display = "none";
      break
    case graphdata:
      graphdata.style.display = "none";
      break
```

```
            }
        });
    }
}

loadMosaicPageHandlers();
function loadMosaicPageHandlers() {
    if (document.body.contains(document.getElementById('imageModal2'))) {
        var createAlbumDropdown = document.getElementById("myDropdown");
        document.getElementById("createAlbumIMG").addEventListener("click", function () {
            //https://www.w3schools.com/howto/tryit.asp?filename=tryhow_css_js_dropdown
            createAlbumDropdown.classList.toggle("show");
        });
        document.getElementById("creatAlbumBTN").addEventListener("click", function () {
            createAlbumDropdown.classList.toggle("show");
            //submit value and then empty value
            var currentpool = document.getElementById("selectedPoolID").value
            var currentMode = document.getElementById("kachelmodeID").value
            var newAlbum = document.getElementById("newAlbumName").value
            xhr.open("GET", "http://localhost:4242/createAlbum?newAlbum=" + newAlbum +
"&currentpool=" + currentpool + "&currentMode=" + currentMode);
            xhr.send();
            document.getElementById("newAlbumName").value = ""

        })
        var images = document.getElementsByClassName("grid-img-MosaicC");
        var modal = document.getElementById("imageModal2");
        var modalImg = document.getElementById("imgModalID");
        Array.prototype.forEach.call(images, function (img) {
            img.onclick = function () {
                //console.log(img.src);
                modal.style.display = "block";
                modalImg.src = this.src;
            }

        });
        var span = document.getElementsByClassName("close")[0];
        span.onclick = function () {
            modal.style.display = "none";
        }
        window.addEventListener("click", function (event) {
            switch (event.target) {
                case modal:
                    modal.style.display = "none";
                    break
            }
        });
        document.getElementById("upload_Btn").addEventListener("click", function () {
            document.getElementById("loadermodal").style.display = "block"
            /*document.getElementById("upload_Btn").disabled = true;*/
            document.getElementById("notePOOL").firstChild.nodeValue = "Bitte ein wenig Geduld, Bild
wird verarbeitet...";
        })
    }
}

loadSettingPageHandlers();
function loadSettingPageHandlers() {
    if (document.body.contains(document.getElementById('profileIDsettings'))) {
        var profilesetting = document.getElementById('profileIDsettings');
        profilesetting.addEventListener("click", function () {
            xhr.open("GET", "http://localhost:4242/settings");
            xhr.send();
        });
        var passwordsetting = document.getElementById('passwordsettings');
        passwordsetting.addEventListener("click", function () {
            xhr5.open("GET", "http://localhost:4242/changePWSite");
            xhr5.send();
        });
        var deleteAccsetting = document.getElementById('deleteACCsettings');
        deleteAccsetting.addEventListener("click", function () {
            xhr5.open("GET", "http://localhost:4242/deleteAccSite");
            xhr5.send();
        });

    }
}


function loadChangeandDeleteHandler() {
    if (document.body.contains(document.getElementById('changePWBtnID'))) {
        document.getElementById("changePWBtnID").addEventListener("click", function () {
            console.log("change PW site")
            var formData = new FormData(document.getElementById("changePasswordForm"));
            xhr5.open('POST', 'http://localhost:4242/changePassword');
            xhr5.send(formData);
        });

    } else if ((document.body.contains(document.getElementById('deleteAccBtnID')))) {
        document.getElementById("deleteAccBtnID").addEventListener("click", function () {
            console.log("delete Account site")
            var formData = new FormData(document.getElementById("deleteAccForm"));
            xhr5.open('POST', 'http://localhost:4242/deleteAccount');
            xhr5.send(formData);
        });

    }
}
  function giveIMGeventHandler() {
    if (document.body.contains(document.getElementById("dropdownDelete"))) {
      var mosaicGalleryDropdown = document.getElementById("dropdownDelete");
      document.getElementById("dropdownOption").addEventListener("click", function () {
        //https://www.w3schools.com/howto/tryit.asp?filename=tryhow_css_js_dropdown
        mosaicGalleryDropdown.classList.toggle("show");
      });

      var albumSelection = document.getElementById("albumSelection");
      albumSelection.addEventListener("change", function () {
        if (document.body.contains(document.getElementById("GALLERY"))) {
          xhr.open("GET", "http://localhost:4242/selectAlbumAndShow?album=" + this.value +
"&page=mosaic");
          xhr.send();
        } else {
          xhr.open("GET", "http://localhost:4242/selectAlbumAndShow?album=" + this.value +
"&page=base");
          xhr.send();
        }
      })


      var deleteAlbumBtn = document.getElementById("deleteAlbum");
      deleteAlbumBtn.addEventListener("click", function () {
        console.log(document.getElementById("albumSelection").value)
        if (document.body.contains(document.getElementById("GALLERY"))) {
          xhr.open("GET", "http://localhost:4242/deleteAlbum?album=" +
document.getElementById("albumSelection").value + "&page=mosaic");
        } else {
          xhr.open("GET", "http://localhost:4242/deleteAlbum?album=" +
document.getElementById("albumSelection").value + "&page=base");
        }
        xhr.send();
      })
    }
    var images = document.getElementsByClassName("grid-img");
    var downloadImg = document.getElementsByClassName("overlayDownload");
    var infoImg = document.getElementsByClassName("overlayInfo");
    var modal = document.getElementById("imageModal");
    var modalImg = document.getElementById("imgModalID");
    var infoModalImg = document.getElementById("imgInfoModalID");
    var infoModalText = document.getElementById("imgInfoText");
    Array.prototype.forEach.call(images, function (img, i) {
      img.addEventListener("click", function () {
        //console.log(img.src);
        modal.style.display = "block";
        modalImg.src = this.src;
        document.getElementsByClassName("deleteIMG")[0].id = img.id;
        document.getElementsByClassName("deleteIMG")[0].onclick = function () {
          console.log("delete " + this.id);
```

```
                xhr.open("GET", "http://localhost:4242/deleteMosaicAndBasic?delete=" + this.id);
                xhr.send();
            };
        });
        downloadImg[i].addEventListener("click", function () {
            console.log(img.src);
        });
        infoImg[i].addEventListener("click", function () {
            infoModalImg.style.display = "block";
            infoModalText.innerHTML = "<br /> Image Information: " + "<br /> <br /> " +
this.getAttribute('title');
        })
    });
    if (document.body.contains(document.getElementsByClassName("close")[1])) {
        var span = document.getElementsByClassName("close")[0];
        span.onclick = function () {
            modal.style.display = "none";
        }
        var span2 = document.getElementsByClassName("close")[1];
        span2.onclick = function () {
            infoModalImg.style.display = "none";
        }

    }



    window.addEventListener("click", function (event) {
        if (!event.target.matches('.dropdownOption')) {
            var dropdowns = document.getElementsByClassName("dropdownDelete");
            var i;
            for (i = 0; i < dropdowns.length; i++) {
                var openDropdown = dropdowns[i];
                if (openDropdown.classList.contains('show')) {
                    openDropdown.classList.remove('show');
                }
            }
        }

        switch (event.target) {
            case modal:
                modal.style.display = "none";
                break
            case infoModalImg:
                infoModalImg.style.display = "none";
                break
        }
    });
}
```

```
window.addEventListener("click", function (event) {
  if (event.target == popUP) {
    popUP.style.display = "none";
  }
});

function createLogin() {
  loadReg_log_popup("loginForm", "LOG IN", "userLogName", "userLogPass", "LOGIN_btn");
}

function createRegister() {
  loadReg_log_popup("registerForm", "SIGN UP", "userRegName", "userRegPass", "SIGNUP_btn");
}

function loadReg_log_popup(formID, formtitle, inputUseName, inputPassName, btn_id) {
  var superparent = createElementID('div', "formparentID")
  var registerForm = createElementID('form', formID)
  var divRegtitl = createElementID("div", "formtitle")
  var titleForm = document.createTextNode(formtitle);
  var divRegName = document.createElement("div");
  var inputRegName = createInput(inputUseName, "text", "", inputUseName + "ID")
  var divRegPW = document.createElement("div");
  var inputRegPW = createInput(inputPassName, "password", "", inputPassName + "ID")
  var divRegButton = document.createElement("div");
  var inputRegButton = createInput("", "button", formtitle, btn_id);
  document.getElementById("feedDIV").innerHTML = "";
  divRegtitl.append(titleForm);
  divRegName.append(document.createTextNode("Username/Email"), inputRegName);
  divRegPW.append(document.createTextNode("Password"), inputRegPW);
  divRegButton.append(inputRegButton);
  registerForm.append(divRegtitl, divRegName, divRegPW, divRegButton);
  var nextarrow = document.createElement('div');
  nextarrow.setAttribute("class", "next round");
  if (formtitle == "LOG IN") {
    superparent.append(registerForm);
    nextarrow.appendChild(document.createTextNode("›"));
    nextarrow.setAttribute("id", "arrowToLogin");
    nextarrow.addEventListener("click", function () {
      createRegister();
    });
    inputRegButton.addEventListener("click", function () {
      console.log("log in")
      var formData = new FormData(document.getElementById("loginForm"));
      xhr.open('POST', 'http://localhost:4242/login');
      xhr.send(formData);
    });
    superparent.appendChild(nextarrow);
    document.getElementById("pupup-contentID").setAttribute("style", "padding: 20px 10px 20px
20px;");
  } else {
```

```
            nextarrow.appendChild(document.createTextNode("‹"));
            nextarrow.setAttribute("id", "arrowToLogin");
            nextarrow.addEventListener("click", function () {
                createLogin();
            });
            inputRegButton.addEventListener("click", function () {
                console.log("Sign UP")
                var formData = new FormData(document.getElementById("registerForm"));
                xhr3.open('POST', 'http://localhost:4242/register');
                xhr3.send(formData);
            });
            superparent.appendChild(nextarrow);
            superparent.append(registerForm);
            document.getElementById("pupup-contentID").setAttribute("style", "padding: 20px 20px 20px
0px;");
        }
        popupContent.replaceChild(superparent, popupContent.childNodes[4]);
    }

    function createElementID(element, idname) {
        var el = document.createElement(element);
        el.setAttribute("id", idname);
        return el;
    }

    function createInput(name, type, value, id) {
        var input = document.createElement('input');
        input.setAttribute("name", name);
        input.setAttribute("type", type);
        input.setAttribute("value", value);
        input.setAttribute("id", id);
        return input;
    }

});
```

3. Quelcode CSS

```
@font-face {
   font-family: silkscreen;
   src: url('fonts/slkscr.TTF');
}

html {
   min-height: 101%;
   margin: 0;
   padding: 0;
}

body {
   margin: 0;
   padding: 0;
   min-height: 100vh;
   height: 100%;
}

li, a {
   font-family: sans-serif;
   font-weight: 500;
   text-decoration: none;
   color: rgb(104, 104, 110);
}

.logo {
   font-family: silkscreen;
   color: rgb(0, 72, 139);
   font-weight: bolder;
   font-size: 35px;
   padding-left: 100px;
}

.logReg {
   font-size: 15px;
   color: white;
   font-weight: bolder;
   padding: 9px 10px;
   background: rgba(0, 136, 169, 1);
   border: 0.8px rgb(71, 71, 71) solid;
   cursor: pointer;
}

.logReg:hover {
   background: cadetblue;
}

a:hover {
   cursor: pointer;
   color: cadetblue;
```

```
}

.logReg:active {
   background: rgb(135, 160, 95);
}

a:active, li:active {
   color: rgb(135, 160, 95);
}

#loginICON {
   width: 11px;
}

#log {
   border-radius: 3px 0px 0px 3px;
   border-right: 0px;
}

#reg {
   border-radius: 0px 3px 3px 0px;
}

.box {
   display: flex;
   flex-flow: column;
   min-height: 100vh;
   height: 100%;
}

.header {
   position: fixed;
   width: 100%;
   z-index: 10;
}

.box .row.content {
   overflow: auto;
   flex: 1 1 auto;
   /*background: rgb(216, 243, 169);*/
   background: rgb(236, 236, 236);
   background: rgb(240, 241, 241);
   padding: 30px 10%;
   margin-top: 51px;
   /*margin: 1px 5%;*/
   height: 100%;
}

.box .row.header {
   box-shadow: 0px 1px rgb(187, 185, 185);
```

```
    flex: 0 1 auto;
    display: flex;
    align-items: center;
    justify-content: flex-start;
    padding: 0px 10px;
    margin-top: 0px;
    background: rgb(255, 255, 255);
    /* background: rgb(216, 243, 169);*/
}

.box .row.footer {
    flex: 0 1 40px;
    text-align: right;
    height: auto;
    align-items: center;
    justify-content: center;
    padding-right: 20px;
}

.links {
    flex: 0.5;
    /* shorthand for: flex-grow: 1, flex-shrink: 1, flex-basis: 0 */
    display: flex;
    justify-content: flex-start;
    padding-left: 10%;
}

.nav_center {
    flex: 1;
    display: flex;
    list-style: none;
    justify-content: flex-start;
    height: auto;
}

li {
    list-style: none;
}

.nav_center li a {
    /* display: inline-block;*/
    padding: 0px 30px;
    /*padding-right:30px;*/
}

.rechts {
    flex: 1;
    display: flex;
    justify-content: flex-end;
    align-items: center;
    Height: 100%;
    padding: 3px;
    margin-right: 100px;
}

#formtitle {
    padding-bottom: 30px;
}

#loginForm, #registerForm {
    margin-top: 10px;
}

#formparentID {
    display: flex;
    flex-direction: row;
    justify-content: center;
    align-items: center;
    padding: 0 10px;
    width: 100%;
}

.close {
    color: #aaaaaa;
    font-size: 28px;
    font-weight: bold;
}

#closeDIV {
    text-align: right;
}

.close:hover, .close:focus {
    color: #000;
    text-decoration: none;
    cursor: pointer;
}

#popUPID {
    box-shadow: 0px 2px 2px rgba(0, 0, 0, 0.1);
    display: none;
    /* Hidden by default */
    position: fixed;
    padding-top: 100px;
    width: 100%;
    height: 100%;
    background-color: rgb(0, 0, 0);
    background-color: rgba(0, 0, 0, 0.7);
}
```

```
.pupup-content {
   /*text-align: center;*/
   font-family: silkscreen;
   background-color: #fefefe;
   margin: auto;
   margin-top: 50px;
   border: 1px solid #888;
   border-radius: 6px;
   width: 350px;
   height: 350px;
   background: rgb(238, 191, 37);
}

#pupup-contentID div {
   margin-bottom: 10px;
}

#pupup-contentID input {
   /*font-family: "Comic Sans MS", cursive, sans-serif;*/
   margin-top: 5px;
   border-radius: 4px;
   border: 0.5px solid rgb(255, 255, 255);
   padding: 3px 3px;
   /* font-size: 15px;*/
   width: 90%
}

#pupup-contentID input:hover, #pupup-contentID input:focus {
   border: 0.5px solid rgb(63, 201, 206);
}

#SIGNUP_btn, #LOGIN_btn {
   font-family: silkscreen;
   font-size: 20px;
   cursor: pointer;
   color: white;
   background: rgb(0, 72, 139);
}

#loginRegIMG {
   position: absolute;
   margin: 0px auto;
   left: 0;
   right: 0;
   top: 60px;
   width: 105px;
   border-radius: 50%;
   background: #60c7c1;
   padding: 15px;
}
```

```
#loginRegIMGDIV, #feedDIV {
   text-align: center;
}

#formtitle {
   text-align: center;
   font-size: 25px;
}

#arrowToLogin {
   margin-right: 20px;
}

.next {
   margin-top: 60px;
   border-radius: 50%;
   background: #60c7c1;
   padding: 8px 16px;
   cursor: pointer;
}

.next:hover {
   background: rgb(177, 235, 170);
}

#logout {
   border-radius: 50%;
   border: 2px blue solid;
   width: 40px;
   height: 40px;
}

#profile {
   display: flex;
   align-items: center;
   justify-content: center;
}

#profile:hover {
   background: rgb(188, 200, 207);
}

.directionColumn {
   flex-direction: column;
}

.directionColumn ul {
   display: none;
   position: absolute;
```

```
  background-color: #f3f3f3;
  min-width: 160px;
  margin-right: 40px;
  top: 50px;
  padding-left: 0px;
  margin-top: 0;
  /*border: 1px dotted rgb(85, 84, 84);*/
  margin-left: 40px;
}

.directionColumn li {
  height: 100%;
  padding: 0;
}

.directionColumn ul li {
  padding: 12px 16px;
  font-size: 15px;
  padding-bottom: 5px;
  border: 1px rgb(85, 84, 84) dotted;
}

.directionColumn ul li:hover {
  background: lightblue;
}

.directionColumn:hover #submenu1 {
  display: block;
}

.submenuIMAGE {
  width: 15px;
  margin-right: 10px;
}

.barIMAGE:hover {
  width: 20px;
}

.barIMAGE {
  width: 18px;
  padding-right: 12px;
}

.siteTitle {
  text-align: center;
  margin-bottom: 15px;
}

.mosaicBasicTitle {
    margin-bottom: 5px;
}

.welcomeTitle {
    margin-bottom: 5px;
    font-size: 3.4vw;
    font-family: silkscreen;
    color: rgb(93, 79, 223);
}

#currentlyNoIMAGESID {
    display: flex;
    align-items: center;
    flex-direction: column;
}

.noImages {
    margin-top: 50px;
    max-width: 80;
    width: 500px;
    opacity: 0.7;
}

.centertext {
    align-items: center;
}

.siteTitle {
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    font-size: 25px;
    color: rgb(5, 5, 5);
}

.grid-containerGallery {
    justify-content: center;
    display: grid;
    grid-template-columns: repeat(3, 33%);
    /* grid-template-columns: repeat(3, 1fr);*/
    grid-auto-rows: 22vw;
    /*grid-template-rows: repeat(4, 20vw);*/
    /*grid-gap: 15px;*/
    /* grid-template-columns: auto auto auto;*/
    padding: 10px;
    grid-column-gap: 15px;
    grid-row-gap: 10px;
}

.grid-item {
    position: relative;
    background-color: rgba(255, 255, 255, 1);
    text-align: center;
    align-content: center;
    justify-content: center;
}

.grid-img, .grid-img-MosaicC {
    width: 100%;
    height: 100%;
    object-fit: cover;
}

.grid-item:hover .grid-img {
    opacity: 0.8;
    /*filter: brightness(85%);*/
    transition: all 0.5s ease;
    cursor: pointer;
}

.grid-item:hover .overlay {
    opacity: 0.95;
}

.overlay {
    position: absolute;
    bottom: 5%;
    right: 5%;
    opacity: 0;
    transition: .3s ease;
    cursor: pointer;
}

.overlayDownload, .overlayInfo {
    width: 25px;
    height: 25px;
}

/* The imageModal (background) */

.imageModal, #imgInfoModalID, #poolModulID, #imageModal2, .loadermodal, #poolGenerator-
Modal, #poolModalshowData, #showGraph-Pool-Img-Modal {
    display: none;
    position: fixed;
    z-index: 11;
    left: 0;
    top: 0;
    width: 100%;
    height: 100%;
    overflow: auto;
    background-color: rgb(0, 0, 0);
    background-color: rgba(0, 0, 0, 0.9);
```

```
}

#avgRGBflex-container {
   margin-top: 5px;
   display: flex;
   margin-left: 60px;
}

.colRect {
   display: inline-block;
   width: 10px;
   height: 10px;
   border-radius: 50%;
   margin-right: 10px;
}

.greenPoly {
   fill: rgba(150, 243, 150, 0.9);
   /*stroke: rgb(95, 94, 94);
stroke-linejoin: round;*/
}

.redPoly {
   fill: rgba(252, 114, 114, 0.9);
}

.bluePoly {
   fill: rgba(115, 115, 245, 0.9);
}

.redColRect {
   background: rgba(252, 114, 114, 0.9);
}

.greenColRect {
   background: rgba(150, 243, 150, 0.9);
}

.blueColRect {
   background: rgba(115, 115, 245, 0.9);
}

#poolModalshowData, #poolGenerator-Modal, #showGraph-Pool-Img-Modal {
   background-color: rgba(0, 0, 0, 0.6);
}

#poolModulID {
   background-color: rgba(0, 0, 0, 0.7);
}
```

```
.loadermodal {
   /* display: none; gute nachricht, stopt animation im hintergrund
https://stackoverflow.com/questions/34869684/does-a-css3-animation-run-when-parent-element-
has-visibility-hidden*/
   top: 100%;
   background-color: rgba(0, 0, 0, 0);
}

/* imageModal Content (image) */

#imgModalID, #loaderModalID {
   margin: auto auto;
   position: fixed;
   top: 0;
   bottom: 0;
   left: 0;
   right: 0;
   max-width: 80%;
   max-height: 80%;
   padding-top: auto;
}

#imgInfoText, #choosePool-modalContent, #poolGenerator-Content-Modal, #graph-PoolModal-
content {
   display: flex;
   margin: auto auto;
   position: fixed;
   background: rgba(255, 255, 255, .7);
   border-radius: 2px;
   top: 0;
   bottom: 0;
   left: 0;
   right: 0;
   width: 50%;
   height: 60%;
   padding: 5px;
   padding-top: auto;
}

#poolGenerator-Content-Modal {
   flex-direction: column;
   background: rgba(255, 255, 255, 1);
   width: 300px;
   height: 240px;
}

text {
   font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}
```

```
#graph-PoolModal-content {
   flex-direction: column;
   background: rgba(255, 255, 255, 1);
   width: 440px;
   height: 450px;
}

.graph-Pool-Title {
   color: gray;
   text-align: center;
   font-size: 20px;
   font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
   padding-bottom: 10px;
   border-bottom: rgb(151, 151, 151) 1px solid;
}

#pool-Graph {
   margin-top: 10px;
   margin-left: auto;
   margin-right: auto;
}

#avgRGBText {
   font-size: 14px;
   font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}

.close {
   position: absolute;
   top: 15px;
   right: 35px;
   color: #f1f1f1;
   font-size: 40px;
   font-weight: bold;
   transition: 0.3s;
}

.deleteIMG {
   width: 50px;
   height: 50px;
   position: absolute;
   bottom: 35px;
   right: 35px;
   cursor: pointer;
}

.close:hover, .close:focus {
   color: #bbb;
   text-decoration: none;
   cursor: pointer;
}

#notePOOL {
   /*font-size: 20px;*/
   color: rgb(0, 153, 255);
   margin-bottom: 10px;
   padding-left: 10px;
}

#gridBoxImgPool {
   margin-top: 50px;
   justify-content: center;
   display: grid;
   grid-template-columns: repeat(2, 1fr);
   grid-auto-rows: 6vw;
   /*grid-gap: 15px;*/
   padding: 10px;
   grid-column-gap: 15px;
   grid-row-gap: 10px;
}

.grid-imgPools-item {
   border-radius: 10px;
   background-color: rgb(123, 210, 231);
   display: flex;
   /*justify-content: center;*/
   cursor: pointer;
   align-items: center;
}

.right-flex, .center-flex {
   height: 100%;
   display: flex;
   justify-content: center;
   align-items: center;
}

.right-flex {
   justify-content: flex-end;
   flex: 1;
}

.center-flex {
   flex: 20;
}

.grid-imgPools-item:hover {
   opacity: 0.9;
}
```

```
#choosePool-modalContent {
   padding: 0px;
   flex-direction: column;
   background: rgba(255, 255, 255);
   width: 300px;
   height: 370px;
}

.pool-scroll-Container {
   padding-top: 12px;
   width: 300px;
   height: 265px;
   overflow-y: scroll;
}

#uploadPool_Btn {
   display: none;
}

#poolModalcreate, #generator_btnDiv {
   color: rgb(51, 51, 51);
   padding-top: 10px;
   padding-left: 20px;
   border-top: rgb(151, 150, 150) solid 1px;
}

#generator_btnDiv {
   padding-top: 0px;
}

#plusCreatePool {
   display: flex;
   align-items: center;
   cursor: pointer;
}

#plusCreatePool:hover {
   color: rgb(8, 73, 116);
}

#poolModalcreate2 {
   display: none;
   color: rgb(51, 51, 51);
   padding-top: 10px;
   padding-left: 20px;
   border-top: rgb(151, 150, 150) solid 1px;
   align-items: center;
}

#poolModalTitle, #poolGenerator-Title {
   text-align: center;
   color: rgb(121, 120, 120);
   padding: 10px;
   border-bottom: rgb(151, 150, 150) solid 1px;
}

#poolGenerator-Title {
   font-size: 20px;
}

.poolChooseDiv {
   color: rgb(77, 76, 76);
   margin-bottom: 5px;
   padding: 1px;
   padding-left: 15px;
}

#addnewPoolID, .kachelPic, .deletePoolIMG, .downloadPoolIMG, .infoPoolIMG, .deleteWholePool {
   width: 20px;
   height: 20px;
   margin-right: 10px;
}

.deletePoolIMG, .downloadPoolIMG, .deleteWholePool {
   cursor: pointer;
}

#newPoolNameTitle {
   font-size: 12px;
}

#createPoolname {
   width: 90%;
   border: none;
   border-bottom: black 1px solid;
}

#createPoolbtnDiv, #generator_submitDiv {
   padding-top: 10px;
   padding-bottom: 10px;
   text-align: right;
   margin-right: 10%;
}

#createPoolbtn, #addToPoolbtn {
   border: none;
   background: none;
   cursor: pointer;
   color: rgb(8, 73, 116);
}
```

```
#createPoolbtn:active, #addToPoolbtn:active {
   border: none;
   color: rgb(197, 140, 34);
}

#poolFeed {
   margin: auto;
   padding-left: 10px;
   margin-bottom: 10px;
}

#poolModal-ContentData {
   display: flex;
   margin: auto auto;
   position: fixed;
   background: rgba(255, 255, 255, .7);
   border-radius: 5px;
   top: 0;
   bottom: 0;
   left: 0;
   right: 0;
   padding: 5px;
   flex-direction: column;
   background: rgba(255, 255, 255);
   width: 600px;
   height: 500px;
}

.pool-modal-title {
   color: rgb(121, 120, 120);
   font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
   padding: 15px;
   text-align: center;
   font-size: 25px;
}

.pooldata-scroll-Container {
   overflow-y: auto;
   /*overflow-y: scroll;*/
   padding: 10px;
   height: 380px;
   padding-top: 20px;
   border-radius: 4px;
   margin-left: 5px;
   margin-right: 5px;
   border: #888 .5px solid;
   box-shadow: none;
}
```

```
.pooldataDiv {
   display: flex;
   flex-direction: row;
}

.deletePoolDIV {
   text-align: right;
   padding-right: 10px;
}

.kachelname {
   width: 400px;
   white-space: nowrap;
   overflow-x: hidden;
   text-overflow: ellipsis;
}

.left-PoolDiv {
   flex: 1 1 auto;
   text-align: right;
}

#previewTitle {
   font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
   margin-top: 40px;
   font-size: 18px;
   padding-left: 60px;
   color: rgba(165, 164, 164, 0.9);
}

#beforeAfterMosaicDiv {
   margin-top: 10px;
   display: grid;
   grid-template-columns: 0.75fr 0.4fr 0.75fr;
   grid-auto-rows: 25vw;
   padding: 10px;
   padding-left: 60px;
   padding-right: 60px;
}

.beforeAfterMosaic {
   display: flex;
   justify-content: center;
   align-items: center;
   width: 100%;
   height: 100%;
   color: rgba(172, 171, 171, 0.5);
   font-size: 60px;
}
```

```
.beforeAfterBorder {
  border-radius: 5px;
  border: 7px rgba(172, 171, 171, 0.5) dashed;
  font-family: silkscreen;
}

.unselectable {
  user-select: none;
}

.grid-img-MosaicC {
  border-radius: 10px;
  /*border: 1px rgba(194, 191, 191, 0.5) solid;*/
  cursor: pointer;
}

.grid-img-MosaicC:hover {
  filter: brightness(80%);
}

@media only screen and (min-width: 900px) {
  .grid-containerGallery {
    justify-content: center;
    display: grid;
    grid-template-columns: repeat(4, 25%);
    /* grid-template-columns: repeat(3, 1fr);*/
    grid-auto-rows: 18vw;
    /*grid-gap: 15px;*/
    padding: 10px;
    grid-column-gap: 15px;
    grid-row-gap: 10px;
  }
  #gridBoxImgPool {
    grid-template-columns: repeat(3, 1fr);
    grid-auto-rows: 3vw;
  }
}

@media only screen and (max-width: 900px) {
  #previewTitle {
    padding-left: 10px;
  }
  .overlay {
    display: none;
  }
  #beforeAfterMosaicDiv {
    grid-auto-rows: 30vw;
    padding-left: 10px;
    padding-right: 10px;
  }
}

.loader {
  border: 8px solid rgba(172, 171, 171, 0.5);
  border-radius: 50%;
  border-top: 8px solid #3498db;
  width: 60px;
  height: 60px;
  animation: spin 2s linear infinite;
}

@keyframes spin {
  0% {
    transform: rotate(0deg);
  }
  100% {
    transform: rotate(360deg);
  }
}

#fieldset-flex-Div {
  display: flex;
}

.flex-span-right {
  flex-grow: 1;
  text-align: right;
  margin-right: 20px;
}

#generator-select-input-Div {
  margin-bottom: 15px;
}

.generator-poolname {
  margin-top: 10px;
}

#generator-size-Div {
  margin-bottom: 50px;
}

#generator-inputsDIV div {
  margin-left: 20px;
}

#showPoolModulIDbtn {
  margin-left: 10px;
}
```

```
#selectAlbumDiv {
   text-align: right;
}

select {
   border-radius: 2px;
   background: white;
}

#albumSelection {
   color: rgb(108, 108, 238);
   background: transparent;
   font-size: 16px;
   border: 0.5px rgb(207, 205, 205) solid;
}

#albumSelection:hover {
   color: rgb(211, 128, 51);
}

#albumSelection option {
   background: transparent;
   background-color: rgba(255, 255, 255, 0.5);
}

#createAlbumIMG {
   width: 17px;
   vertical-align: middle;
   padding-bottom: 3px;
   cursor: pointer;
}

#chooseAlbumID {
   border: none;
   background: transparent;
}

#albumMosaicSpan {
   background: white;
   border: rgb(164, 164, 168) 1px solid;
   padding: 1px;
   border-radius: 2px;
   margin-right: 4px;
}

.dropdown {
   position: relative;
   display: inline-block;
}
```

```
.dropdown-content {
   display: none;
   margin-top: 2px;
   position: absolute;
   background-color: #ffffff;
   box-shadow: 0px 8px 16px 0px rgba(0, 0, 0, 0.2);
   border-radius: 2px;
   border: rgb(155, 155, 158) 1px solid;
   padding: 2px;
   z-index: 1;
   width: 180px;
   height: 50px;
}

.displayFlex {
   display: flex;
   flex-direction: column;
   margin: 2px;
}

#newAlbumnameDIV {
   margin-bottom: 2px;
}

#createAlbumBtnDIV {
   margin: 3px;
   text-align: right;
}

#creatAlbumBTN {
   font-size: 14px;
   font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
   color: rgb(94, 94, 233);
   border: none;
   background: none;
   cursor: pointer;
}

#creatAlbumBTN:hover {
   color: rgb(233, 171, 36);
}

.dropdownOption {
   padding-left: 3px;
   padding-right: 3px;
   font-size: 17px;
   color: rgb(94, 94, 233);
   border-radius: 20%;
   border: 0.5px rgba(154, 154, 155, 0.5) solid;
   cursor: pointer;
}

.dropdownOption:hover {
   color: rgb(233, 171, 36);
}

.dropdownDelete {
   display: none;
   /* https://stackoverflow.com/questions/22519377/css-dropdown-menu-with-submenu-aligning-
to-the-right-edge-of-its-parent
https://www.w3schools.com/css/tryit.asp?filename=trycss_dropdown_right*/
   left: auto;
   right: 0;
   position: absolute;
   z-index: 11;
   background: rgb(236, 236, 236);
   margin-top: 2px;
   padding: 2px;
   border-radius: 2px;
   border: 0.5px rgba(189, 189, 192, 0.5) solid;
   font-size: 14px;
}

#deleteAlbum {
   white-space: nowrap;
   font-size: 13px;
   font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
   color: rgb(104, 104, 161);
   cursor: pointer;
}

#deleteAlbum:hover {
   color: rgb(233, 171, 36);
}

.show {
   display: block;
}

.centerContantDiv {
   background-color: white;
   margin: auto;
   max-width: 1000px;
   min-width: 80%;
   Height: 500px;
   border: 1px solid rgb(75, 77, 75);
   border-radius: 5px;
   padding: 0px;
}
```

```
.sidenav {
   background-color: rgb(117, 116, 116);
   width: 200px;
   height: 100%;
   margin: 0px;
}

.sidenav ul {
   width: 100%;
   height: 100%;
   margin: 0px;
   padding: 0px;
}

.sidenav ul li {
   margin: 0px;
   color: #f1f1f1;
   list-style: none;
   padding: 15px 20px;
   border-bottom: 1px solid rgba(209, 208, 208, 0.3);
}

.sidenav ul li:hover {
   cursor: pointer;
   color: #639eeb;
}

.flexRow {
   display: flex;
   height: 100%;
}

#settingWelcomeTitle, .settingTitle {
   margin-top: 20px;
   text-align: center;
   font-size: 25px;
}

#settingWelcomeText {
   margin-top: 40px;
   text-align: center;
}

#settingsContent {
   display: flex;
   flex-direction: column;
   width: 80%
}

#helloIcon, .iconKeyDelete {
  width: 25px;
}

.settingstitle {
  margin-left: 140px;
}

.centerForm {
  margin-top: 40px;
}

.cdForm {
  display: flex;
  flex-direction: column;
  margin: auto;
  width: 300px;
}

.cdForm input {
  margin: 2px;
  font-size: 14px;
}

.feedbackstring {
  text-align: center;
  margin-top: 5px;
  font-size: 14px;
}
```