Entwicklung eines webbasierten ePortfolios

im Rahmen einer Umschulung zum Fachinformatiker - Anwendungsentwicklung

Moritz Mandler & Didier Zielke

7. Juni 2020

Prüflingsnummer:
XXX.XXX.XXX

Ausbildungsbetrieb:
BFW Berufsförderungswerk Hamburg
Marie-Bautz-Weg ???
22159 Hamburg

Ausbilder:
Frau Treubel
Herr Dr. Kubillus

Inhaltsverzeichnis

1 Einführung

Die IT-Solution & Design GmbH ist eine Lernfirma innerhalb des Berufsförderungswerks Hamburg GmbH (BFW). Als norddeutsches Zentrum für berufliche Rehabilitation und Integration ist das BFW Hamburg kompetenter Partner für Unternehmen, Träger der beruflichen Rehabilitation und Versicherungen und vor allem für Menschen, die aus gesundheitlichen Gründen ihre bisherige Tätigkeit nicht mehr ausüben können. Es soll eine einfache Plattform zur Veröffentlichung einer 'Mappe' der eigenen Arbeitsergebnisse geboten werden. Benutzer müssen sich authentifizieren und können dann die eigene Mappe bearbeiten. Innerhalb dieser Mappe, die hier nur eine Webseite darstellt, können beliebige Download-Dateien, Texte und Bilder eingefügt werden. So soll den Benutzern auch die Möglichkeit gegeben werden Lebensläufe, Arbeitsproben oder Zertifikate auf der eigenen Seite abzulegen. Die IT-Solutions & Design hat vom BFW Hamburg den Auftrag erhalten diese Webanwendung zu erstellen.

2 Projektdefinition

2.1 Ist-Analyse

Die Teilnehmer*innen des BFW müssen sich während ihrer Ausbildung auf einen Praktikums- und einen Arbeitsplatz bewerben. Viele Bewerbungen werden dabei per E-Mail versendet. Um eine E-Mail nicht zu groß werden zu lassen, werden dabei manchmal Arbeitsergebnisse, Zertifikate o. ä. nicht mit gesendet. Das könnte den Gesamteindruck im Bewerbungsprozess verschlechtern. Außerdem müssen die Teilnehmer*innen entscheiden, auf welche Unterlagen, Arbeitsproben und Bilder sie verzichten wollen.

2.2 Anforderungsdefinition (Soll-Konzept

Wie im Pflichtenheft angegeben (siehe Anhang) sollen folgende Anforderungen erfüllt werden:

i. Musskriterien

- Neuanlegen, Ändern und Löschen von Benutzern durch den Admin
- Jedem Benutzer ist eine Benutzerseite zugeordnet, die nur von dem jeweiligen Benutzer bearbeitet werden darf
- Neuanlegen, Ändern und Löschen von Texten (auch Links) auf der jeweiligen Benutzerseite
- Hochladen von Bildern und Pdf-Dateien innerhalb der Benutzerseite
- Berücksichtigung verschiedener Berechtigungsstufen (Admin, Benutzer, Gast)
- Nur registrierte Gäste dürfen die Inhalte der Benutzer einsehen. Anlegen eines Gasts über Formular mit Überprüfung der E-Mailadresse (Freischaltung des Gastaccounts über E-Mail-Link) und Versenden eines generierten Kennworts

ii. Wunschkriterien

- Gäste dürfen nur die Portfolios sehen, welche ihnen über eine Freigabe zugeordnet wurden.

iii. Abgrenzungskriterien

- Keine Überprüfung auf Verletzung von Urheberrechten oder Verstöße gegen das Datenschutzgesetz

3 Projektplanung

3.1 Ressourcenplanung

Als Ressource sollen ein PC mit der im Pflichtenheft angegebenen technischen Produktumgebung zur Verfügung gestellt werden. Für die Kalkulation wurde eine Stundenplanung gemacht. Zwei Personen sollen dem Projekt für die folgenden Arbeitsschritte zur Verfügung stehen.

Software-Entwurf	
Use-Case-Diagramm	4h
Klassenmodell	4h
Datenmodell	4h
Summe:	12h

Realisierung	
Entwurf der Testfälle	4h
Programmierung Geschäftslogik	24h
Programmierung Datenzugriffsklassen	10h
Programmierung Controller-Klassen	24h
Programmierung View-Klassen	20h
Summe:	82h

Tests	
Testfälle programmieren und durchführen	10h
Eventuelle Fehlerbeseitigung	4h
Summe:	

Abschluss	
Soll-Ist-Vergleich	6h
Dokumentation	25h
Übergabe	1h
Summe:	32h
Gesamtsumme:	**140h**

3.2 Kosten

Die Kosten für dieses Projekt belaufen sich bei einem Stundensatz von 35,00€ und einem Gesamteinsatz von 140Std. pro Person, wie aus der Ressourcenplanung hervorgeht, auf:

2 Personen * 140Std * 35,00€ = **9800€**

4 Projektdurchführung

4.1 Softwareentwurf

Zuerst werden die Anwendungsfälle ermittelt. Es wird drei Hauptanwendungsfälle geben. Die Nutzung als Administrator, als Benutzer und als Gast. Wir entscheiden uns für eine zentrale Login-Oberfläche und danach zu einer strikten Trennung der Hauptanwendungsfälle.

Aus Gründen der Einfachheit wird im Klassenmodell und im Datenmodell zwischen den Benutzergruppen Administrator, Benutzer und Gast über eine Eigenschaft status unterschieden, ansonsten jedoch das gleiche Model genutzt. Es ist darauf zu achten, dass eine mögliche Doppelbelegung der Kombination E-Mail und Passwort vorkommen könnte. In diesem Fall kann das Login System keine eindeutige Zuordnung durchführen. Dieser Fall ist sehr unwahrscheinlich und wird daher nur für den Fall das ein Gast von mehreren Portfolios Berechtigungen mit dem gleichen Passwort erhalten hat, behandelt.

Für die Umsetzung der Wunschkriterien wird die **Benutzeroberfläche** unterteilt in:

- Ansichten aller Seiten des eigenen Portfolios mit Bearbeitungsmöglichkeit
- Möglichkeit zum erstellen und löschen neuer Seiten innerhalb des eigenen Portfolios
- Gästeverwaltung zum erstellen und löschen von Gästen des eigenen Portfolios und setzten derer Berechtigungen

Für die Umsetzung der Wunschkriterien wird die **Gastoberfläche** unterteilt in:

- Ansicht aller freigegebenen Seiten eines Gastes
- Downloadmöglichkeit der hinterlegten Dateien auf freigegebenen Seiten

Für die Umsetzung der Wunschkriterien wird die **Adminoberfläche** unterteilt in:

- Möglichkeit zum erstellen neuer Benutzer
- Übersicht aller existierenden Benutzer und Gäste mit Lösch-Option
- Möglichkeit zum erstellen neuer Administratoren
- Übersicht aller existierenden Administratoren mit Lösch-Option

All diese Hauptanwendungen sind über eine Navigationsleiste verfügbar.

Damit keine Karteileichen in der Datenbank entstehen, wird beim Löschen eines Gastes stets auch jede seiner Berechtigungen aus der Datenbank entfernt. Beim Löschen eines Users werden automatisch auch all seine Berechtigungen, seine Gäste und die Berechtigungen derer aus der Datenbank entfernt.

Bei allen Formularen wurde aus Sicherheitsgründen darauf geachtet keine Datenbank Id oder Ähnliches zur Identifizierung auf der Serverseite zu übergeben. Stattdessen wird ein Index in ein Array übergeben, welches lediglich erlaubte Aktionen zulässt. So ist keine Form-Manipulation bei beispielsweise dem löschen von Gästen möglich.

Ein weiterer Schwerpunkt ist die Sicherheit der Daten eines jeden Benutzers. Wir entschieden uns aus Gründen der performance nicht für die Speicherung von Dateien in der Datenbank.

Wenn bei dem Prinzip der Objektorientierung eine strikte Trennung der Zuständigkeiten von Klassen und Methoden beachtet wird, ist die Wartbarkeit des Codes einfacher. Hier ist ein einfaches MVCModell geplant, bei welchem die Model-Klassen die Geschäftslogik implementieren, die View-Klassen die Ein- und Ausgaben darstellen und die Controller-Klassen für die Verbindung dazwischen eingesetzt werden sollen.

Für jeden Anwendungsfall soll es mindestens eine Controller-Klasse geben, die über den Frontcontroller aufgerufen wird. Außerdem soll es Klassen geben, die für die Schnittstelle der ankommenden HTTP-Anfragen zuständig sind (Request) und Klassen, die für die Antwort benutzt werden (Response). Neben den reinen Modelklassen wird geplant, Datenzugriffsklassen zu implementieren, die für die Datenbankzugriffe (CRUD) verwendet werden sollen. Durch die Verwendung eines DAO-Interfaces könnte man schnell die Datenzugriffsklassen für andere Datenbanksysteme erweitert werden. Aus Zeitgründen wurde aber hierauf verzichtet.

4.2 Daten- und Klassenentwurf

Das Klassenmodell (Abbildung 3) und das Datenmodell (Abbildung 2) befindet sich im Anhang auf den Seiten 16 und 16.

Der nach dem Login validierte Nutzer wird als Objekt der Klasse User inklusive der dazugehörigen Seiten-Liste in eine Session Variable geschrieben und dient im weiteren Verlauf der Anwendung zur Identifikation sowie zur Legitimierung. Ein weiterer Vorteil ist, dass so ein schneller und bequemer Zugriff auf die wichtigsten User-Daten jederzeit gewährleistet ist.

Da ein Benutzer mehrere Seiten in seinem Portfolio haben kann, wurde hier eine 1:n Beziehung im Datenmodell genutzt. Da eine Seite sich aus mehreren Inhalten zusammensetzen kann, wird auch hier eine 1:n Beziehung im Datenmodell genutzt. Da ein Gast keine eigenen Seiten hat, wird eine Berechtigungs-Tabelle für Seiten der Benutzer genutzt. Ein Gast kann mehrere Berechtigungen haben, aber jede Berechtigung muss eindeutig einer Person zugeordnet sein. Daher wird hier eine 1:n Beziehung im Datenmodell genutzt.

Im Klassenmodell wird festgelegt, dass jeder User eine Liste von Seiten beinhaltet. Jede Seite beinhaltet wiederum eine Liste von Inhalten.

Jeder User wird definiert über eine eindeutige Id (Primärschlüssel), vorname, nachname , E-Mail und Passwort, sowie einen status der entweder admin, user oder guest heisst. Jeder User enthält zudem eine Liste von Seiten, welche, wenn vorhanden, definiert werden über eine eindeutige Id (Primärschlüssel), die Id des Seitenerstellers und den Titel der Seite. Jede dieser Seiten enthält zudem eine Liste von Inhalten, welche jeweils Dateinamen und/oder html/text enthalten.

Die Liste an Seiten eines jeden Users wird anhand der Berechtigungstabelle der Datenbank initialisiert. Bei einem Admin wird diese Liste also leer sein.

Damit für einen existierenden Gast oder Benutzer immer mindestens eine Seite mit Inhalt existiert, die nach dem Login angezeigt werden kann, entscheiden wir uns für eine Standard Home Seite. Diese Seite kann jeder Benutzer selbst verändern und gestalten, jedoch nicht löschen. Diese Seite wird außerdem all seinen Gästen zur Begrüßung angezeigt, sprich die Berechtigungen werden automatisch jedem Gast gewährt und können nicht entzogen werden.

4.3 Realisierung

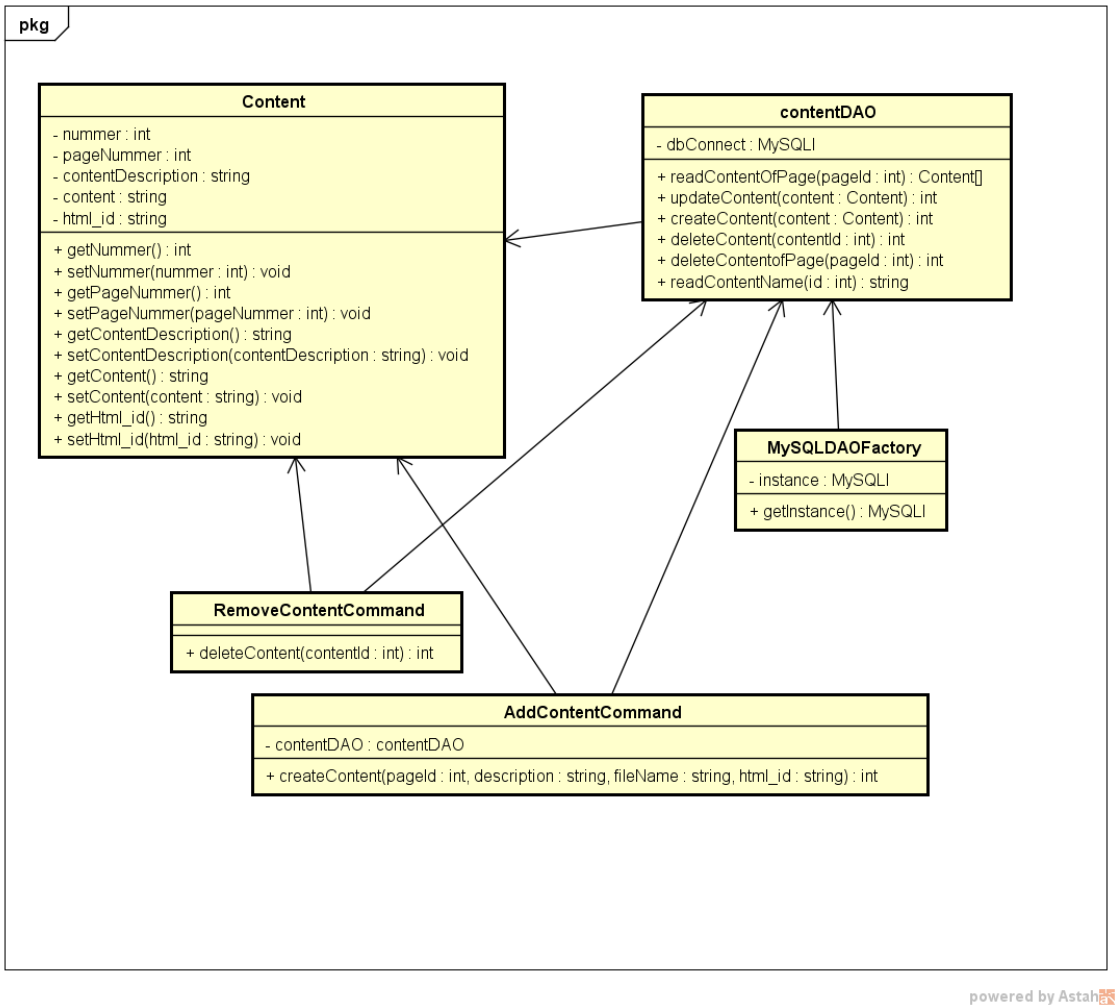
4.3.1 Übersicht

Im Entwurf wurde die Entscheidung für das MVC-Muster getroffen. Das Modell ist dabei für die Geschäftslogik und die Persistenz verantwortlich, der Controller nimmt die Benutzereingaben entgegen, wertet sie aus und übergibt Ergebnisse an die entsprechende Präsentations-Schicht. Der View (Präsentation) ist für die Interaktion zum Benutzer verantwortlich. Hier werden die Eingaben des Benutzers entgegen genommen und die Darstellung der Ausgaben getätigt.

4.3.2 Programmierung der Geschäftslogik und der Datenzugriffsklassen

Die Klassen User, Page und Content werden gemäß dem Pflichtenheft umgesetzt. Zusätzlich werden Klassen konzipiert, in denen die einzelnen Anwendungsfälle wie Seiten des Gastes anzeigen oder Seiten des Users bearbeitenentwickelt werden. Über die Datenzugriffsklassen UserDAO und PageDAO und ContentDAO gibt es die Möglichkeit Daten aus der Datenbank zu lesen oder in die Datenbank zu schreiben. Aus Vereinfachungsgründen ist in der unten stehenden Abbildung nur der Mapper ContentDAO angegeben. Für die Datenbankkommandos gibt es aber eine ähnliche UserDAO sowie eine PageDAO.

Abbildung 1: Geschäftslogik



Eine Datenbankverbindung erhalten die Datenmapper UserDAO, PageDAO und ContentDAO über das Singleton MySQLDAOFactory. Ein Singleton ist ein Entwurfsmuster, welches eine statische Eigenschaft enthält, hier die Verbindung zur Datenbank. Auf diese Eigenschaft kann über eine statische Methode zugegriffen werden. Dabei ist wichtig, dass es nur eine Instanz/ ein Objekt dieser Klasse geben darf.

So werden alle Methoden, die etwas mit der Datenbank zu tun haben, über die Eigenschaft dbConnect aufgerufen. Der Austausch zwischen den Datenzugriffsklassen und den übrigen Komponenten wird über die Standardklassen User, Page und Content gemacht. Entweder wird ein Objekt vom Typ Content übergeben, um einen neuen Inhalt (Content) anzulegen (siehe: Schreiben von neuen Datensätzen) oder es wird zum Lesen ein neues Objekt vom Typ Content angelegt und an die aufrufenden Methode zurückgegeben (siehe: Lesen von Datensätzen).

Lesen von Datensätzen:

```
16 public function readContentOfPage($pageNr){
17         $contentList = array();
18         $pageNummer = "".$pageNr;
19         $sql = "SELECT content.*
20                 FROM content
21                 WHERE content.page = ?
22                 ORDER BY content.id";
23         $preStmt = $this->dbConnect->prepare($sql);
24         $preStmt->bind_param("s", $pageNummer);
25         $preStmt->execute();
26                 $preStmt->store_result();
27         $preStmt->bind_result($id, $page, $content, $description,
              $html_id);
28
29         while($preStmt->fetch()){
30                         $thisContent = new Content($id, $page,
                              $description, $content, $html_id);
31
32                         $contentList[] = $thisContent;
33         }
34
35         $preStmt->free_result();
36         $preStmt->close();
37
38         return $contentList;
39     }
```

Schreiben von Datensätzen:

```
103 public function createContentByCopy(Content $content){
104
105         $pageNummer = (int)$content->getPageNummer();
106         $contentDescription = "".$content->getContentDescription()
              ;
107         $thisContent = "".$content->getContent();
108         $html_id = "".$content->getHtml_id();
109
110         $ok = -1;
111         $sql = "INSERT INTO content (page, content, description,
              html_id)
112                 VALUES (?,?,?,?)";
113         $preStmt = $this->dbConnect->prepare($sql);
114         $preStmt->bind_param("ssss", $pageNummer, $thisContent,
              $contentDescription, $html_id);
115         $preStmt->execute();
116         $ok = $preStmt->insert_id;
117         $preStmt->close();
118         unset($preStmt);
119
120         return $ok;
121     }
```

4.3.3 Programmierung der Controllerklassen

Über das Front-Controller-Pattern (Anhang D) wird eine zentrale Stelle geschaffen, die Anfragen bearbeitet und an die betreffenden Controller-Klassen weiterleitet. Die Command-Control-Schicht hat die Aufgabe eine Anforderung (Request) zu analysieren und die Anfrage mit allen Parametern an den Teil der Anwendung weiterzuleiten, der für die Verarbeitung verantwortlich ist. So erfüllt im Command-Control-Pattern jede Command-Instanz eine spezielle Aufgabe. Eine bestimmte Command-Instanz bearbeitet den Fall, dass vorhandener Content auf Seiten entfernt werden soll (RemoveContentCommand), und eine andere, wenn neuer Content erstellt werden soll (CreateContentCommand). Innerhalb der Command-Instanz wird die Verarbeitung an die Businesslogik weitergeleitet, die die Daten verarbeitet, speichert und eventuell ein Ergebnis zurückliefert. Diese Ergebnisse werden als Antwort (Response) an den Browser zurückgeschickt. Eingaben werden mittels Request an den Server weitergeleitet. Sämtliche Details zu den Anfragen sind in den beiden Super Globals *$_REQUEST* und *$_SERVER* gespeichert. Unabhängig von der verwendeten Methode werden alle Name-Werte-Paare in *$_REQUEST* zur Verfügung gestellt. Dies ist ein assoziatives Array, welches in allen PHP-Scripten sichtbar ist. Das heißt, dass eine URL *index.php?cmd=RemoveContent* innerhalb des Array *$_REQUEST* mit *$_REQUEST['cmd']* den Wert *RemoveContent* liefert. Zuerst werden die Klassen zur Speicherung der Request-Parameter geschrieben. Das Interface Request definiert die Methoden:

- public function getParameterNames();
- public function issetParameter($name);
- public function getParameter($name);
- public function getHeader($name);

Wobei in der konkreten Klasse HttpRequest das ganze *$_REQUEST*-Array in der Eigenschaft parameters gespeichert wird, so dass über die Methode getParameter der Wert des einzelnen Eintrags zurückgegeben wird.

```
1  public function getParameter($name){
2      if (isset($this->parameters[$name])) {
3          return $this->parameters[$name];
4      }
5      return null;
6  }
```

Mit textit$cmdName = $request->getParameter("cmd"); wird der Name des nächsten Commands ermittelt. Und mit *$command = $this->loadCommand($cmdName);* die entsprechende Command-Klasse geladen.

Für die verschiedenen Command-Klassen wird wieder ein Interface angelegt, um sicherzustellen, dass jede Command-Klasse die gleiche Methode implementiert. Das Interface enthält nur eine Methodendefinition:

```
1  interface Command{
2      public function execute(Request $request, Response $response);
3  }
```

In jeder Command-Implementierung steht der Code, der hier ausgeführt werden soll. Der FrontController lädt die entsprechende Command-Instanz auf und führt genau diese Methode auf.

```
1  public function handleRequest(Request $request, Response $response
       ){
2      $command = $this->resolver->getCommand($request);
```

```
3       $command->execute($request, $response);    $response->flush();
4   }
```

Das Interface Response dient den Ausgaben, die dem Benutzer nach der Ausführung der entsprechenden Command-Instanz angezeigt werden sollen. In der konkreten Implementierung HttpResponse werden alle Daten, die geschrieben werden an die Eigenschaft body gehängt:

```
1   public function write($data){
2       $this->body .= $data;
3   }
```

So können innerhalb der Command-Instanzen die Ausgaben mit response->write() getätigt werden. Für die Antwort an den Client dient die Methode flush, die alle Inhalte überträgt und die Variablen wieder leert:

```
1   public function flush(){
2       header("HTTP/1.0 " . $this->status);
3       foreach ($this->headers as $name => $value){
4           header($name . ":" . $value);
5       }
6       print $this->body;
7       $this->headers = array();
8       $this->data = null;
9   }
```

4.3.4 Programmierung der Viewklassen

Darstellung der Dateien Die Seiten und Inhalte eines Benutzers bzw. die ein Gast einsehen darf werden als Arrays im jeweiligen Objekt abgelegt. Diese werden mit *foreach* verarbeitet und abhängig von MIME-Type unterschiedlich dargestellt:

PDF-Dokumente	Link, der einen neuen Tab im Browser öffnet, in welchem die PDF dargestellt wird
Audio-Dateien	Direkt auf der Seite abspielbar
Video-Dateien	Direkt auf der Seite abspielbar
Alle weiteren	Downloadlink

Um den unberechtigten Zugriff zu verhindern, werden die Dateien außerhalb des Webroots gespeichert. Damit der Pfad über den Browser nicht ersichtlich ist, wird die Datei in einen base64-String umgewandelt und als URI in den HTML-Code eingefügt.

Als Beispiel befindet sich der Code der UserHome View im Anhang (Listing 1, Seite 13)

4.3.5 Dateiverwaltung

Dateiupload Beim Upload von Dateien durch den Benutzer wird zunächst geprüft, ob es sich auch tatsächlich und eine hochgeladene Datei handelt. Aus Sicherheitsgründen werden danach die MIME-Types der Dateien geprüft und mit einem Array zulässiger Typen verglichen. So wird vermieden, dass der Benutzer Shell-Scripts, php-Scripts usw. hochlädt, selbst wenn er die Dateiendung ändert. Sollte die Datei zulässig sein, wird geprüft, ob bereits eine Datei mit gewählten Namen existiert und ggf. eine Meldung ausgegeben und der Upload abgebrochen. Ist diese Prüfung negativ wird geprüft, ob der Dateiname zulässig ist. Nach aktuellem Stand ist nur der Dateiname "defaultContentünzulässig. Wenn der Name zulässig ist, wird die Datei in den Ordner des Users verschoben und der Dateiname in der Tabelle *content* eingetragen.

Benutzerordner Jeder Bentuzer erhält beim Anlegen einen eigenen Ordner, der nach seiner ID benannt wird. In diesem werden seine Uploads gespeichert. Aus Sicherheitsgründen befindet sich der Benutzerordner außerhalb des Webroots um einen ungewollten Zugriff von Außen zu verhindern.

> **Alternative:**
>
> Es kann auch in den Tabellen *user*, *content* und *page* jeweils eine weitere Spalte hinzugefügt werden mit einer zufälligen Folge aus Buchstaben und Zahlen, die eine externe ID darstellen sollen. Dies erhöht nochmals die Sicherheit, da manche Daten über GET in der URL übermittelt werden.

Löschen von Dateien Will der Benutzer seinen Content löschen, so wird nicht nur der Datesatz aus der Datenbank gelöscht, sonder auch die Datei. Der Benutzer kann somit sicher sein, dass seine Dateien nicht auf ungewollt versteckt verbleiben. Hierzu wird zunächst ermittelt welches Betriebssystem auf dem Server läuft um mit dem korrekten Befehl die Datei zu entfernen:

```
53 if(strtoupper(substr(PHP_OS, 0, 3)) === 'WIN') {
54     unlink(USERS_DIR . $currentUser->getId() . '/' . $file);
55 } else {
56     shell_exec('rm ' . USERS_DIR . $currentUser->getId() . '/' .
          $file);
57 }
```

Verhalten bei Accountlöschung Möchte der Benutzer seinen Account vollständig löschen, so werden alle seine Dateien sowie sein Benutzerordner vom System gelöscht. Um dies sicherzustellen wird auch hier zuerst das Betriebssystem ermittelt. Bei Windows wird zunächst jede Datei einzeln gelöscht um anschließend den Ordner zu entfernen. Bei Linux Systemen sowie BSD und anderen unixoiden System kann der Ordner samt Inhalt mit einem an die Shell übergebenem Befehl gelöscht werden:

```
31 if(strtoupper(substr(PHP_OS, 0, 3)) === 'WIN') {
32     foreach (glob(USERS_DIR . $userId . "/*.*") as $filename) {
33         unlink($filename);
34     }
35     rmdir(USERS_DIR . $userId);
36 } else {
37     shell_exec('rm -rf ' . USERS_DIR . $userId);
38 }
```

Als Beispiel befindet sich der vollständige Code zur Accountlöschung durch den Admin im Anhang (Listing 2, Seite 14).

4.4 Tests

Test	Ergebnis
Selber Gast von zwei Benutzern	Der Gast kann sich mit zwei verschiedenen Passwörtern anmelden
s. o. mit gleichem Passwort	Gast kann sich die Portfolios aussuchen, die er sehen will
Anmeldung ohne Validierung	nicht möglich, Gast muss seine Adresse erst validieren

5 Soll-/Ist-Vergleich

Folgende Musskriterien wurden implementiert:

- Neuanlegen, bearbeiten und löschen von Benutzern durch den Admin
- Jeder Benutzer hat eine Seite, die nur er bearbeiten kann
- Anlegen, bearbeiten und löschen von Texten und Links
- Upload von Dateien (Bilder, Dokumente)
- nur registrierte Gäste mit validierter Adresse können die Portfolios einsehen.

Folgende Wunschkriterien wurden implementiert:

- Gäste können nur für sie freigegebene Portfolios einsehen

Es konnten alle Kriterien implementiert werden. Der Code ist erweiterbar und lässt weitere Implementierungen zu, wenn der Kunde dies wünscht.

6 Fazit

Die Entwicklung des ePortfolios ermöglicht einen ersten Einblick in den Aufbau und Struktur von Social Media Seiten. Einer der wichtigsten Aspekte bei der Entwicklung stellt die Sicherheit dar und die jeweiligen Tests um diese zu erhöhen. Bei der Entwicklung ist uns aufgefallen, dass man sehr viele Optionen implementieren kann um den User größtmögliche Individualität zu ermöglichen. Auch beim Design gibt es sehr viel Spielraum, der ein erfahrener Webdesign nutzen kann um den Benutzer ein optisch ansprechendes Portfolio zu bieten, mit welchem er sich bei Kunden und potenziellen Arbeitgeber präsentieren kann.

7 Anhang

7.1 Code Listings

Listing 1: UserHome.php

```
1  <!DOCTYPE html>
2  <html lang="en">
3      <head>
4          <meta charset="UTF-8">
5          <meta name="viewport" content="width=device-width, initial-scale=1.0">
6          <title>Document</title>
7          <link href="./css/<?= $this->style ?>.css" rel="stylesheet" type="text
               /css">
8          <link rel="shortcut icon" type="image/x-icon" href="./favicon_96x96.
               png" size="48x48">
9      </head>
10     <body>
11         <div id="header">
12             Das ePortfolio von <?= $this->displayname ?>!
13         </div>
14         <ul>
15             <?php
16                 $indexOfPageList = 0;
17                 foreach($this->pageList as $page) {
18                     echo '<li><a href="./index.php?cmd=UserHome&page='.
                           $indexOfPageList.'">'. $page->getTitle() ."</a></li
                           >";
19                     $indexOfPageList++;
20                 }
21             ?>
22             <li><a href="./index.php?cmd=AddPage">ś</a></li>
23             <li style="float: right"><a href="index.php?cmd=Logout">Logout</a
                   ></li>
24             <li style="float: right"><a href="./index.php?cmd=UserSettings">
                   Einstellungen</a></li>
25         </ul>
26         <div id="main">
27
28             <h2><?= $this->requestedTitle ?></h2>
29             <?= $this->alert ?>
30             <?= $this->editLink ?><br>
31             <?= $this->addContentLink ?>
32
33             <?php
34
35                 $output = "";
36                 foreach($this->requestedContent as $content) {
37
38                     if(!$content->getContent() == null) {
39                         $file = $this->filepath . $content->getContent();
40
41                         if(file_exists($file)) {
42
43                             $mimeType = mime_content_type($file);
44
45                             $tmp = base64_encode(file_get_contents($file));
46
47                             if(!strstr($mimeType, "image") == false) {
48                                 $output = '<a href="data:'.$mimeType.';base64
                                       ,'.$tmp.'" target="_blank"><img src="data
                                       :'.$mimeType.';base64,'.$tmp.'"></a><br><
                                       br>';
49                             } else if(!strstr($mimeType, "audio") == false) {
```

```
50                                  $output = '<audio controls><source src="data
                                        :'.$mimeType.';base64,'.$tmp.'" type="'.
                                        $mimeType.'">Der Browser unterstütz das
                                        Format '.$mimeType.' nicht!</audio><br><br
                                        >';
51                              } else if(!strstr($mimeType, "video") == false) {
52                                  $output = '<video controls><source src="data
                                        :'.$mimeType.';base64,'.$tmp.'" type="'.
                                        $mimeType.'">Der Browser unterstütz das
                                        Format '.$mimeType.' nicht!</video><br><br
                                        >';
53                              } else {
54                                  $output = '<a href="data:'.$mimeType.';base64
                                        ,'.$tmp.'" target="_blank">'. $content->
                                        getContent() .'</a><br><br>';
55                              }
56                          }
57                      } else {
58                          $output = "";
59                      }
60                      echo "<div id='content'>" . $output . $content->
                            getContentDescription() . "<br></div>";
61                  }
62              ?>
63          </div>
64          <footer>&copy; 2020 M. Mandler & D. Zielke</footer>
65      </body>
66  </html>
```

Listing 2: RemoveUserCommand.php

```
1   <?php
2       namespace classes\commands;
3
4       use classes\request\Request;
5       use classes\response\Response;
6       use classes\template\HtmlTemplateView;
7       use classes\mapper\UserDAO;
8       use classes\model\User;
9
10      class RemoveUserCommand implements Command{
11          public function execute(Request $request, Response $response) {
12
13              if (isset($_SESSION['admin'])){
14                  $currentUser = unserialize($_SESSION['admin']);
15              } else {
16                  header('location: index.php');
17              }
18              if (!($currentUser instanceof User)){
19                  header('location: index.php');
20              }
21              if ($currentUser->getStatus() != "admin"){
22                  header('location: index.php');
23              }
24
25              $userDAO = new UserDAO;
26
27              if($request->issetParameter('deleteUser')) {
28                  $userId = $request->getParameter('deleteUser');
29                  $userDAO->deleteUser($userId);
30
31                  if(strtoupper(substr(PHP_OS, 0, 3)) === 'WIN') {
32                      foreach (glob(USERS_DIR . $userId . "/*.*") as $filename)
                            {
```

```
33                          unlink($filename);
34                      }
35                      rmdir(USERS_DIR . $userId);
36                  } else {
37                      shell_exec('rm -rf ' . USERS_DIR . $userId);
38                  }
39              }
40
41              $listOfAllUsers = $userDAO->readAllUsersWithPages("user");
42              $multiListArray = array();
43              for($i = 0; $i < count($listOfAllUsers); $i++){
44                  $multiListArray[$i][] = $listOfAllUsers[$i];
45                  $multiListArray[$i][] = $userDAO->readGuestListOfUser(
                        $listOfAllUsers[$i]->getId());
46              }
47
48              $view = 'RemoveUser';
49              $template = new HtmlTemplateView($view);
50              $style = "default"; // provisorisch
51              $template->assign('style', $style);
52              $template->assign('userList', $multiListArray);
53              $template->render( $request, $response);
54          }
55      }
56  ?>
```

7.2 Abbildungen

Abbildung 2: Datenmodell

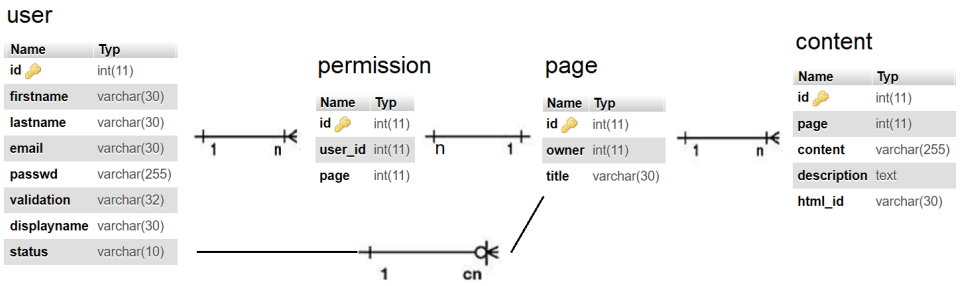


Abbildung 3: Klassendiagramm

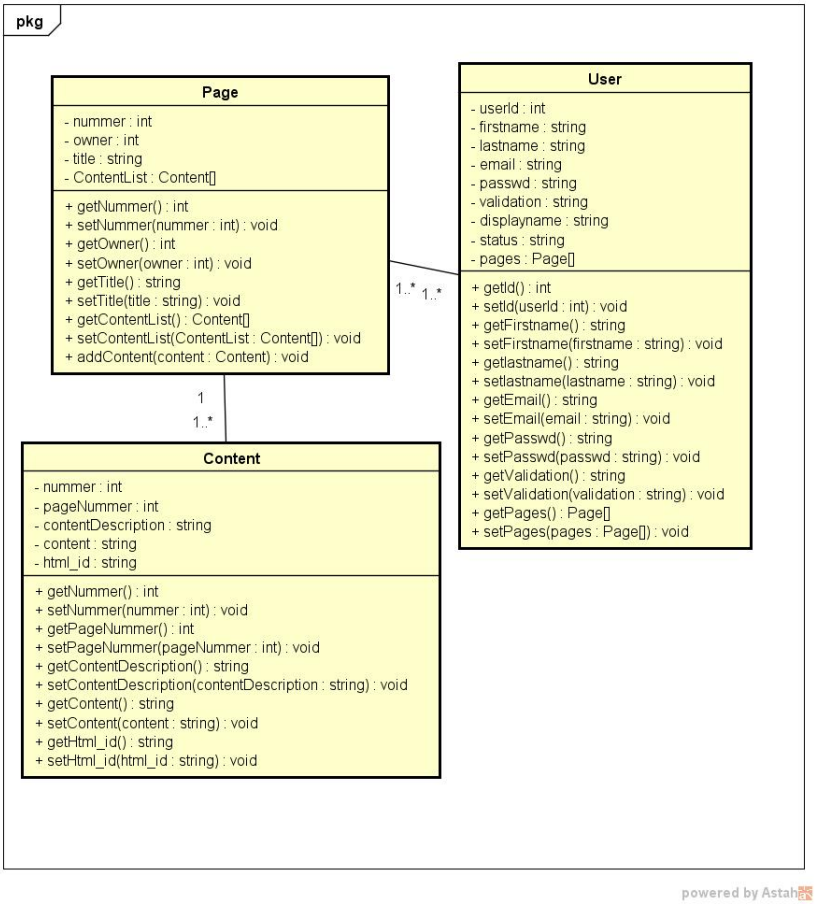


Abbildung 4: Use Case Diagram