Webservice - Frontend

Projekt: DigitalSchoolNotes

Projekt Team: Adler, Brinnich, Hohenwarter, Karic, Stedronsky

Version 5.0 **24.02.2016** **Status: [RELASE]**

	Datum	Name	Unterschrift
Erstellt	30.10.2015	Niklas Hohenwarter	
Geprüft	24.02.2016	Niklas Hohenwarter	
Freigegeben			
Git-Pfad: */doc/technologien*		**Dokument:** *webservice_frontend_technologie.doc*	

Inhaltsverzeichnis

1 Changelog

Version	Datum	Status	Bearbeiter	Kommentar
0.1	2015-10-30	Erstellt	Niklas Hohenwarter	Dokumentation erstellt
0.2	2015-11-04	Bearbeitet	Philipp Adler	Pagination hinzugefügt
1.0	2015-11-04	Geprüft	Thomas Stedronsky	Rechtschreibfehler
1.1	2015-11-25	Bearbeitet	Thomas Stedronsky	Text-Element hinzugefügt
1.2	2015-11-25	Bearbeitet	Adin Karic	Code-Element hinzugefügt
2.0	2015-11-25	Geprüft	Niklas Hohenwarter	QA
2.1	2015-12-15	Bearbeitet	Selina Brinnich	Codeverzeichnis und Abbildungsverzeichnis hinzugefügt
2.2	2015-12-16	Bearbeitet	Selina Brinnich	Jquery Draggable hinzugefügt
2.3	2015-12-16	Bearbeitet	Adin Karic	Codeelement überarbeitet
2.4	2015-12-16	Bearbeitet	Philipp Adler	Heftelement hinzugefügt
3.0	2015-12-16	Geprüft	Niklas Hohenwarter	QA
3.1	2016-01-20	Bearbeitet	Thomas Stedronsky	Textelement erweitert
3.2	2016-01-20	Bearbeitet	Philipp Adler	Bildelement hinzugefügt
3.3	2016-01-21	Bearbeitet	Adin Karic	Codeelement erweitert
4.0	2016-01-21	QA	Niklas Hohenwarter	QA
4.1	2016-02-18	Bearbeitet	Adin Karic	OCR hinzugefügt
4.2	2016-02-22	Bearbeitet	Thomas Stedronsky	PWS hinzugefügt
5.0	2016-02-24	QA	Niklas Hohenwarter	QA

2 Routing

Um einzelne Zustände in der Applikation über eine URL aufrufen zu können, wird ein ui-router [1] benötigt. Dieser ermöglicht es, aufgrund einer URL einen Controler zuzuweisen. Desweiteren basiert routing nicht mehr nur auf URLs sondern auf states. Das bedeutet, dass der Benutzer durch z.B. einen Klick auf einen Button in einen anderen Status weitergeleitet wird. Die Seite verändert sich ohne neu geladen zu werden. Es folgt ein Beispiel einer Route:

```
mainApp.config(function($stateProvider, $urlRouterProvider,
$locationProvider, $httpProvider) {

    $urlRouterProvider.otherwise('/'); // Falls keine andere route zutrifft

    // MAIN PAGE
    $stateProvider.state('mainpage', { //Name des Stati
        abstract: true, // Kann nicht direkt instanziert werden (parent)
        url: '', // Da keine URL angegeben wurde ist es /
        templateUrl: '/mainpage/mainpage.html', //HTML Template
        controller: 'mainpageCtrl', // Für die View zuständiger Controller
        data: {
            authorization: false, // Muss man angemeldet sein?
            admin: false // Muss man admin sein?
        }
    });

});
```

Code 1: Beispiel der Konfiguration einer Route

3 Forms

Es können Forms mit HTML erstellt werden. Es ist jedoch zusätzlich möglich diese Forms zu validieren. Als erstes muss ein Form definiert werden:

```
<form ng-submit="submitRegister()" method="POST" name="register" novalidate
>
```

- ng-submit: Definiert welche Methode beim drücken des Submit Buttons aufgerufen werden soll
- novalidate: Deaktiviert die HTML5 Form validierung

Nun ein Feld inklusive Validierung:

```
<div class="form-group" style="padding-top:15px;" ng-class="{ 'has-error':
register.email.$dirty && register.email.$invalid }">
```

Das ng-class ändert die CSS Klasse des Objektes falls die Bedingung nach dem Doppelpunkt zutrifft. $dirty überprüft ob der User schon etwas im Feld verändert hat und $invalid prüft ob die Eingabe im Feld valide ist.

```
      <p  style="color: red;" ng-show="(submitted || register.email.$dirty)
&& register.email.$error.required">Bitte gib deine E-Mail Adresse an.</p>
      <p  style="color: red;" ng-show="(submitted || register.email.$dirty)
&& register.email.$error.pattern">Bitte gib eine valide E-Mail Adresse
an.</p>
```

Hier werden Fehler angezeigt falls die Bedingung in ng-schow zutrifft. Submitted wird auf true gesetzt sobald das Form einmal “abgeschickt” wurde.

```
      <p style="color: red;" ng-show="emailerror">{{ email_error }}</p>
```

Hier wird ein Error aus Django ausgegeben.

```
      <input class="form-control" name="email" type="text" style="height:
50px;" placeholder="E-Mail Adresse*" ng-model="email" ng-pattern="/[a-z0-
9!#$%&'*+=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+=?^_`{|}~-]+)*@(?:[a-z0-9](?:[a-
z0-9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?/" required/>
</div>
```

ng-model definiert eine Variable im $scope von Javascript. Dadurch kann man im Controller mittels $scope.email auf den Inhalt des Feldes zugreifen. Ng-pattern validiert den Inhalt des Feldes mittels einer REGEX.

In Javascript sieht das wie folgt aus:

```
$scope.submitRegister = function(){ // Wird im Form aufgerufen
    $scope.submitted = true; // Damit die Val Errors angezeigt werden
    var email = $scope.email; // Holt den Wert aus dem Input Feld
    $scope.emailerror = false; // Init
    $scope.email_error = ''; // Init
    if(!$scope.register.$valid) { //Falls im Form keine Fehler mehr sind
        $http({
            method: 'POST',
            url: '/api/register', //Poste an URL
            headers: {'Content-Type': 'application/json'}, // Als JSON
            data: {email: email} // mit den Attributen
        })
            .success(function (data) {
                if (data['registration_error'] != null) {//Error aus Django
                    $scope.emailerror = true; // Error da
                    $scope.email_error = data['registration_error'];
                }else{
                    alert('Vielen Dank für deine Registrierung!\n' +
                        'Sobald du deine E-Mail Adresse bestätigt hast
kannst du dich einloggen und sofort starten!'); // success message falls ok
                }
            })
            .error(function (data) {

            });
    }
}
```

Code 2: Validieren und Versenden von User-Daten

Sobald das Form valide ist, werden die Daten an den Server geschickt. Dort werden sie aus Sicherheitsgründen nochmals validiert. Es wird überprüft ob die Email-Adresse schon von einem User verwendet wird. Falls ja, wird ein Fehler zurück gegeben, welcher dann angezeigt wird.

4 Captcha

Wie das Captcha letzendlich validiert wird, ist im Backend Dokument beschrieben. Um das Captcha anzuzeigen muss eine JS Lib eingebunden werden[2].

```
<script
src="https://www.google.com/recaptcha/api.js?onload=vcRecaptchaApiLoaded&re
nder=explicit" async defer></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/angular-
recaptcha/2.2.5/angular-recaptcha.min.js"></script>
```

Dannach kann das Captcha einfach wie folgt ins Form eingebunden werden:

```
<div vc-recaptcha key="publicKey"></div>
```

publicKey definiert den Public Key des Recaptchas. Dieser wird im Javascript file über $scope zugewiesen. Das Captcha wird nun angezeigt. Um zu überprüfen ob es ausgefüllt wurde kann man folgendes tun:

```
if(vcRecaptchaService.getResponse() === ""){
    $scope.captchaerror = true;
    $scope.captcha_error = "Bitte löse das Captcha.\n";
}
```

Wenn das Captcha gelöst wurde, kann man damit das Form genau ein Mal absenden. Falls es nochmal abgesendet werden muss, muss das Captcha zurückgesetzt werden. Dies kann man wie folgt tun:

```
vcRecaptchaService.reload();
```

5 Pagination

Falls eine Menge an Daten angezeigt werden muss, ist es besser dies über mehrere Seiten zu verteilen. Das wird Pagination genannt. Für die Realisierung benötigt man die Anzahl der Objekte die insgesamt ausgegeben werden sollen. Dann wird definiert, wieviele Elemente pro Seite angezeigt werden sollen. Um die Seitenanzahl zu berechnen, werden alle Elemente durch die Anzahl der Elemente dividiert, die pro Seite dargestellt werden sollen. Der Server übergibt dann die Elemente, welche auf einer Seite angezeigt werden sollen. Falls der User auf eine andere Seite wechselt, werden die nächsten Objekte vom Server bezogen.

```
$http({
        method: 'GET',
        url: '/api/admin_user',
        data: {}
    })
        .success(function (data) {
            $scope.users = data['test'];
            $scope.len = data['len'];
            $scope.currentPage = 0;
            $scope.l = Math.ceil($scope.len/$scope.itemsPerPage);
        })
        .error(function (data) {
        });
    var searchMatch = function (haystack, needle) {
        if (!needle) {
            return true;
        }
        return haystack.toLowerCase().indexOf(needle.toLowerCase()) !== -1;
```

```
};

$scope.range = function (size, start, end) {
    var ret = [];
    if (size < end) {
        end = size;
        start = size;
    }
    for (var i = start; i < end; i++) {
        ret.push(i);
    }
    return ret;
};

$scope.firstPage = function () {
    $scope.currentPage = 0;
};

$scope.prevPage = function () {
    if ($scope.currentPage > 0) {
        $scope.currentPage--;
    }
};

$scope.nextPage = function () {
    if ($scope.currentPage < $scope.l- 1) {
        $scope.currentPage++;
    }
};

$scope.lastPage = function () {
    $scope.currentPage = $scope.l-1;
};

$scope.setPage = function () {
    $scope.currentPage = this.n;
};
```

Code 3: Umsetzung von Pagination

6 Heftelemente

6.1 Heftelemente

Abbildung 1: Elemente hinzufügen

Im Heft kann mittels Mausklick auf die Toolbar ein Element hinzugefügt werden. Dieses Element wird standardgemäß links oben positioniert. Um das Element bearbeiten zu können, wird beim Überfahren des Elements ein Bleistift eingeblendet, welcher dem Benutzer erlaubt den bestehenden Inhalt des Elements zu editieren. Neben dem Bleistift erscheint ebenfalls ein Mistkübel, mit dem das Element entfernt werden kann. Falls die letzte Heftseite angelangt wurde, kann mittels Mausklick auf

den Plus-Button eine neue Heftseite hinzugefügt werden. Dem Benutzer ist es selbst überlassen aus wievielen Seiten sein Schulheft besteht.

7 Textelement

Um Informationen im Sinne von Text in ein Heft einzufügen wird ein Textelement benötigt. Mit diesem Textelement lässt sich Text eingeben und bearbeiten. Es können Funktionen wie Fett, Kursiv, Durchgestrichen, Liste, Aufzählung, etc. auf den Text angewendet werden. Außerdem können Tabellen in dem Textelement erzeugt werden, welche beliebig groß definiert werden können. Sollte der Benutzer Sonderzeichen einbinden wollen geht dies mit dem Sonderzeichen Tool im Textelement. Außerdem kann ein Hyperlink eingebunden werden, der den Link automatisch auf einen neuen Tab weiterleitet.

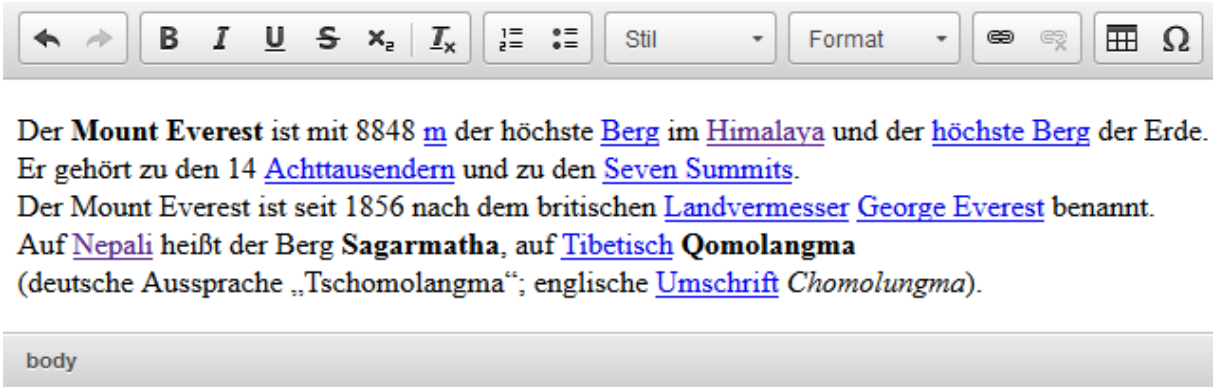

Abbildung 1 Texteditor im Bearbeitungsmodus

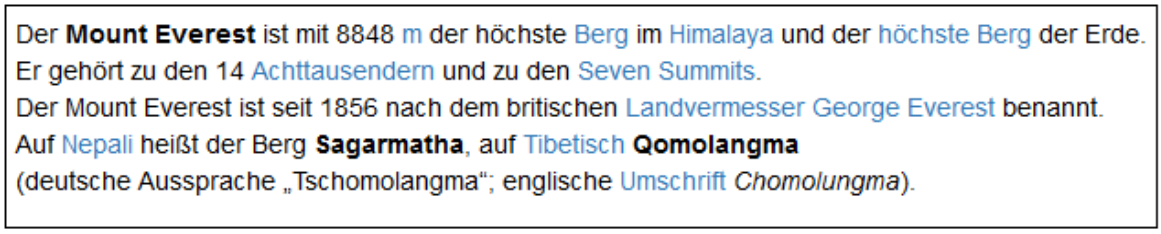

Abbildung 2 Texteditor im Ansichtsmodus

```
<!-- TEXT ELEMENT -->
<span ng-if="element.art == 'textarea'" style="word-wrap: break-word;">
    <!-- Edit -->
    <textarea ng-if="editMode == element.id"
              ng-model="models[element.art][element.id]" ckeditor>
    </textarea>
    <!-- Read-Only -->
    <div ng-if="editMode != element.id"
         style="border: 1px solid black; min-height: 20px; min-width: 20px;
text-align: left; padding:5px;">
        <div ng-bind-html="models[element.art][element.id]" style="word-
wrap: break-word;"></div>
    </div>
</span>
```

Code 4: Einbinden des Texteditors

7.1 Speicherung

Die Speicherung beim Textelement hat sich anfangs als schwierig erwiesen. Allerdings wurde dies dann mithilfe der “on-Events” gelöst. Diese Events lösen nach einen bestimmten Ereignis aus und führen daraufhin einen Befehl bzw. eine Methode aus. In unserem Fall wird bei den Events:

- dataReady
- key
- paste
- selectionChange

```
ck.on('dataReady', updateModel);
ck.on('key', updateModel);
ck.on('paste', updateModel);
ck.on('selectionChange', updateModel);
```
Code 5 Speicherung beim Textelement

Durch diese Anpassung werden die vorgenommenen Formatierungen von Bearbeitungsmodus in den Ansichtsmodus übernommen.

7.2 Anzeige

Es musst außerdem geregelt werden, dass das Textelement im Bearbeitungsmodus immer eine bestimmte Größe anhand des Fenster hat. Dies wurde wie folgt gelöst:

```
width: window.innerWidth * 0.58
```
Code 6 Fensterbreite Textelement

Um das Textelement seitlich nicht aus dem Dokument fahren zu lassen wurde ein Word-Break in den Style eingebaut. Dies wird beim Bearbeitungs –und Ansichtsmodus verwendet.

```
style="word-wrap: break-word;"
```
Code 7 word-wrap

8 Codeelement

Zur Realisierung des Codeelements wird das Framework codemirror [3] benutzt. Das Codeelement soll es dem Benutzer ermöglichen auf einfache und intuitive Weise Codesnippets in seiner Mitschrift zu erstellen und zu bearbeiten. Besondere Features sind hier das Anzeigen der Zeilennummern sowie das Syntax-Highlighting (abhängig von der jeweiligen Programmiersprache). In der vorliegenden Implementation werden folgende Sprachen unterstützt:

- XML, HTML
- JavaScript
- C-ähnliche Sprachen (C, C++, C#, Java, etc.)
- Python
- SQL
- Shell
- Ruby
- PHP

Wenn der Benutzer nun konkret ein Codeelement hinzufügen will, betätigt er zunächst diesen Code-Button.

Abbildung 2 Code-Button

Nach Betätigen des Code-Buttons erscheint nun eine Auswahl von (Programmier-)Sprachen. Hier legt der Beneutzer fest wie der Inhalt im Codeelement gehighlighted wird. (Dies ist von Sprache zu Sprache anders und kann auch im Nachhinein geändert werden)

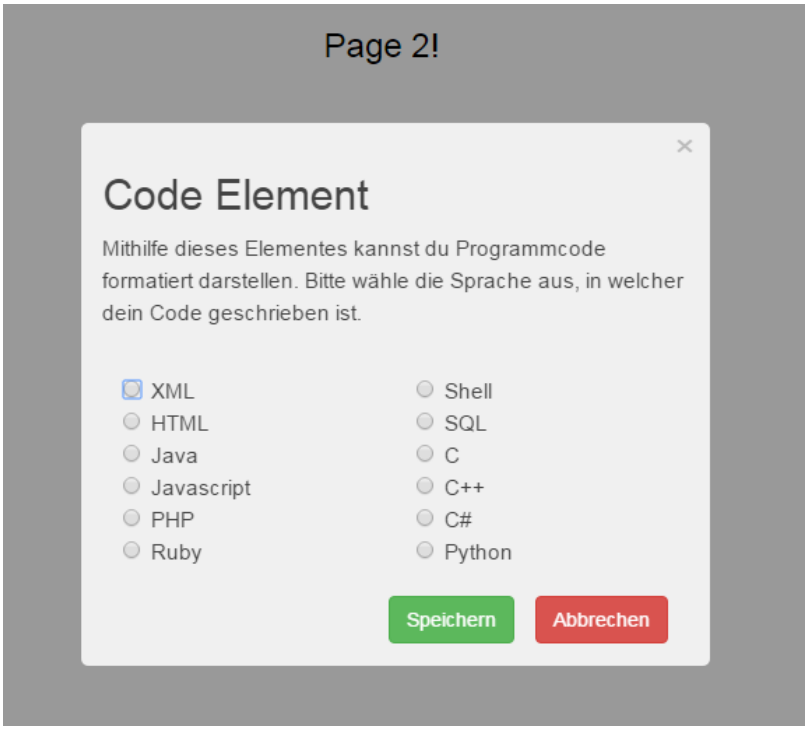

Abbildung 3 Programmiersprache auswählen

Es erscheint anschließend das (leere) Codelement im Heft.

Abbildung 4 Code Element

Unter dem Codeelement gibt es noch drei Buttons die nur sichtbar sind wenn man mit der Maus darüber fährt. Mit Druck auf den ersten Button (also den Mülleimer) kann das Element (nachdem man nochmals bestätigt hat) gelöscht werden. Um in den Bearbeitungsmodus des Codeelements zu wechseln betätigt der User den zweiten Button (also den Stift). Um im Nachhinein die Auswahl der Programmiersprache zu ändern (siehe Dialog oben) muss auf den dritten Button, also das Zahnrad, geklickt werden.

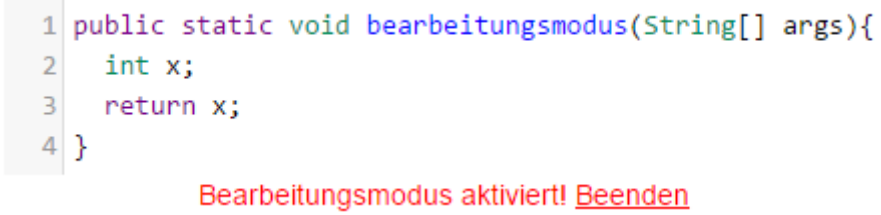

Abbildung 5 Code Bearbeitung

Bei Klick auf den Code-Button wird die Methode codeElementCreate() ausgeführt.

Dort wird ein ngDialog mit dem codeElementSettings-template geöffnet wo der User zum ersten Mal die Programmiersprache auswählen kann. Nachdem dies ausgewählt wurde wird die Methode addCodeElement() ausgeführt.

```
$scope.addCodeElement = function(){

    if(!$scope.wf) {

        data = "{\"data\":\"\", \"language\":\"" + $scope.codeLanguage +
"\"}";
        console.log(data);

        $scope.addelement('code', data);

    }else{

        $scope.wf = false;

        single_object = $filter('filter')($scope.content, function (d)
{return d.id === $scope.cid;})[0];

        data = "{\"data\":\""+single_object['data']['data']+"\",
\"language\":\"" + $scope.codeLanguage + "\"}";

        $scope.editelement($scope.cid, 'code',data);

    }
}
```

Code 8 Codeelement-Persistierung

Bei Klick auf das Zahnrad kann dann noch die eingestellte Programmiersprache geändert werden. Dabei wird die Methode setCodeElementLanguage() ausgeführt.

9 Bildelement

Für die Speicherung der Bilder des Bildelements verwenden wir S3 von Amazon. Das boto s3 Interface (Python API für die Amazon Webservices) bietet Funktionen für das Erstellen, Bearbeiten und Löschen von Dateien auf S3 an. Mit einem Dialog wird der/die Benutzer gefragt, welches Bild und wie groß dieses sein soll.

Abbildung 6 Bild hochladen

Beim Hochladen wird der S3 Link des Bildes mit der übergebenen Größe in die Datenbank gespeichert und anschließend auf der Heftseite ausgegeben.

```
$scope.uploadFile = function(){

          alert($scope.width);

          var file = $scope.myFile;

          console.log('file is ' );

          console.dir(file);

          var uploadUrl = "/api/notebook/upload";

          var message = fileUpload.uploadFileToUrl(file, uploadUrl);

          message.then(function(data) {

              data_data = "{\"data\":\""+data+"\",
\"width\":\""+$scope.width+"\", \"height\":\""+$scope.height+"\"}";

              $scope.addelement('image', data_data);

          });

     };
```

Code 9 Upload Funktion

Im weiteren Verlauf lässt sich die Größe ändern. Auch dafür wird dem User ein Dialog zur Verfügung gestellt, wo von ihm/ihr die neuen Werte abverlangt werden.

```
$scope.editPictureElement = function(){

        data_data =
"{\"data\":\""+$scope.models['image'][$scope.idimage][0]+"\",
\"width\":\""+$scope.width+"\", \"height\":\""+$scope.height+"\"}";

        $scope.editelement($scope.idimage, 'image', data_data);

        $scope.idimage = null;

    }
```

Code 10 Bildelement bearbeiten

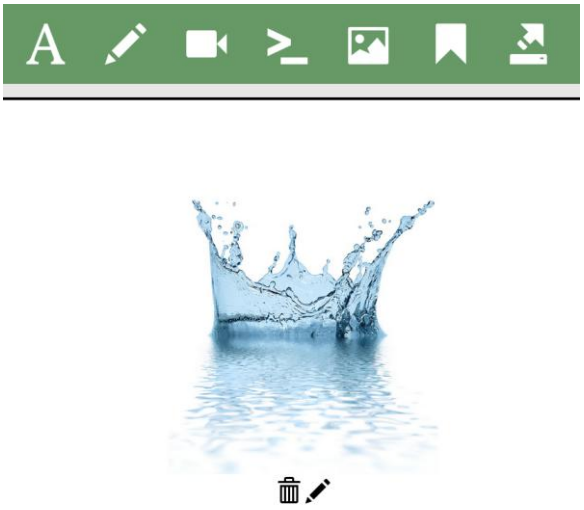

Abbildung 7 Darstellung des Bildelements

10 jQuery Draggable

Um einzelne Elemente innerhalb eines Heftes beliebig verschieben und anordnen zu können, wird das jQuery Draggable Widget [4] verwendet. Dieses ermöglicht es, Elemente einer bestimmten Class oder mit einer bestimmten ID zu selecten und verschiebbar zu machen, sodass der Benutzer das entsprechende Element mit der Maus innerhalb eines definierten Bereiches verschieben kann.

Innerhalb der Applikation hat jedes Heft-Element eine eindeutige ID. Diese setzt sich aus dem Typ des Elementes und der ID für dieses Element zusammen. Mithilfe dieser eindeutigen ID kann ein Draggable Widget für dieses Element erstellt werden:

```
$scope.makeDraggable = function (id, art) {
    $timeout(function(){
        angular.element("#"+art+"_"+id).draggable({
            containment: '#notebook',
            stop: function () {
                var finalPos = $(this).position();
                $scope.editPositionElement(id, art, finalPos.left,
finalPos.top);
            },
            create: function () {
            }
        });
    });
};
```

Code 11: Initialisierung eines Draggable Widgets

Dabei ist darauf zu achten, dass der AngularJS-Service $timeout im Controller hinzugefügt wird. Dieser Service dient dazu, dass der hier definierte Code erst ausgeführt wird, nachdem alle Elemente ins Heft geladen wurden, da der Selector sonst kein entsprechendes Element findet und damit die Elemente nicht verschiebbar wären. Das optionale Attribut *containment* dient zum Definieren des Bereiches, innerhalb dessen Elemente verschoben werden können. In diesem Fall ist das ein div mit der ID „notebook“. Außerdem können unterschiedliche Events definiert werden. In obigem Beispiel wurde nur für das Event *stop* ein eigener Ablauf definiert. Sobald ein Element wieder losgelassen

wird und damit das Verschieben beendet wurde, wird hier die finale Position des Elements ausgelesen und abgespeichert.

Um ein Draggable Widget zu späterer Zeit wieder zu entfernen, gibt es folgende Möglichkeit:

```
$scope.makeUndraggable = function (id, art) {
    angular.element("#"+art+"_"+id).draggable("destroy");
};
```

Code 12: Entfernen eines bestehenden Draggable Widgets

Hier wird ebenfalls ein Selector erstellt und anschließend die Funktion *.draggable()* aufgerufen. Wird dieser Methode nur „destroy“ übergeben, wird das Draggable Widget entfernt. Wird dieser Methode „disable“ stattdessen übergeben, wird das Verschieben vorübergehend deaktiviert. Beim vorübergehenden Deaktivieren wird das entsprechende Element grau hinterlegt. Zum erneuten Aktivieren wird der String „enable“ übergeben.

11 Parallel Working System

Mit dem Parallel Working System ist es dem Benutzer möglich Hefte mit anderen Benutzern zu teilen und diese dann gleichzeitig zu bearbeiten. Grundsätzlich ist es möglich jedes Heft mit beliebig vielen Nutzern zu teilen, hierbei ist nicht relevant ob das Heft öffentlich oder privat ist. Allerdings kann immer nur ein User an einem Element arbeiten. Wenn ein Benutzer gerade an einem Element arbeitet wird allen anderen Nutzern ein rotes Rufzeichen über dem Element angezeigt. In diesem Fall kann das Element weder gelöscht, bearbeitet oder verschoben werden.

Um ein Heft mit andern Benutzern zu teilen muss in die Hefteinstellung navigiert werden, dies funktioniert wie folgt:

Abbildung 8 Hefteinstellungen

Im Textfeld können Usernames eingegeben werden, außerdem werden User zum Hinzufügen vorgeschlagen. Mit dem Minus neben den bereits hinzugefügten User können die Bearbeitungsrechte wieder entzogen werden. Nach diesen Vorgängen muss der Button „Änderungen speichern“ geklickt werden um die Änderungen zu speichern.

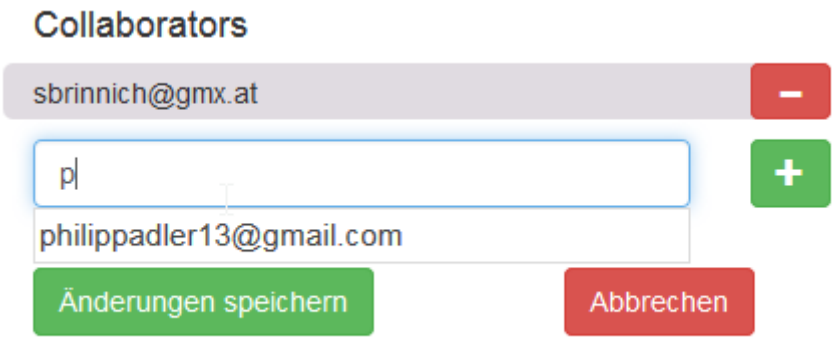

Abbildung 9 Collaborator hinzufügen

Ist ein Heft mit einem Benutzer geteilt, wird es in der gesammelten Heftanzeige unter „für mich freigegeben Hefte“ angezeigt.

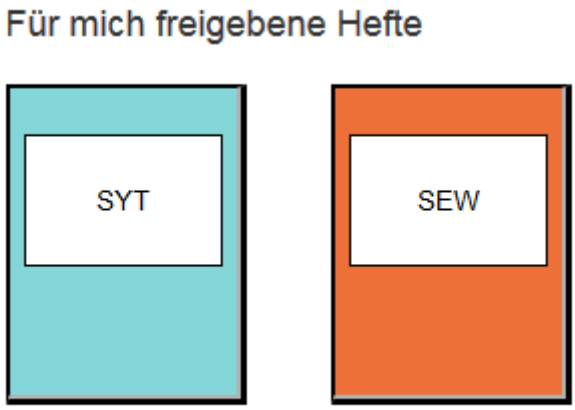

Abbildung 10 freigegebene Hefte

Anschließend können Elemente beliebig bearbeitet werden. Wird gerade an einem Element gearbeitet sieht man ein rotes Rufzeichen über dem Element. Dieses Element kann wie bereits erwähnt nicht bearbeitet, gelöscht oder verschoben werden.

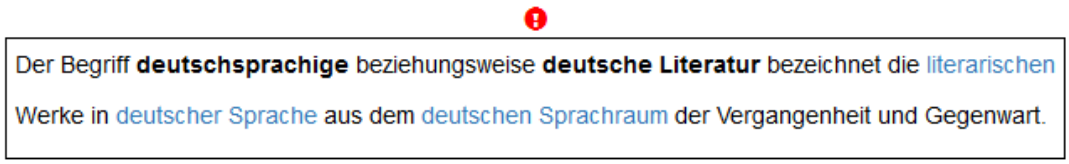

Abbildung 11 Bearbeitungsmodus aktiv

In der Tabelle NotebookContent wurden zusätzliche Attribute hinzugefügt.

```
is_active = BooleanField(default=False)
is_active_by = EmailField()
```
Code 13 NotebookContent Attribute

Hier werden die active User in einem Heft zurückgeliefert.

```
def view_get_is_active(request):
    if not request.user.is_authenticated():
        return JsonResponse({})
    if request.method == "POST":
        notebook = Notebook.objects.get(id=request.POST.get('notebook'))
        findnotebook = None
        content = notebook.content
        for item in content:
            if str(item["id"]) == str(request.POST.get('content_id')) and
item["art"] == request.POST.get('content_art'):
                findnotebook = item
                break
        return JsonResponse({"active":  findnotebook.is_active,
"active_by": findnotebook.is_active_by})
```
Code 14 is_active python

Im *notebookedit.js* wird dies dann überprüft und jedes Mal beim Verlassen beziehungsweise beim Öffnen den Bearbeitungsmodus überprüft und anschließend gesetzt.

Wird der Bearbeitungsmodus verlassen wird das Attribut auf `false` gesetzt.

```
$scope.editelement(id, art, {"data": $scope.models[art][id][0]},false);
```
Abbildung 12 edit_mode false

Sollte der Bearbeitungsmodus aktiviert werden wird das Attribut `is_active` auf true gesetzt.

```
$scope.editelement(id, art, {"data": $scope.models[art][id][0]},true);
```
Abbildung 13 edit_mode true

Außerdem wird gespeichert welcher User gerade aktiv ist in dem jeweiligen Element. Dies wird in `is_active_by` abgespeichert.

Das Prinzip des Parallel Working Systems beruht auf Polling. Es wird alle 10 Sekunden der neue Stand des Contents gepullt. Nach diesem Poll wird entschieden ob sich der Heftinhalt aktualisieren muss. Es wird hierbei überprüft ob es seit dem Letzen Pull Vorgang eine Veränderung gab. Außerdem wird überprüft ob derzeit der zu Aktualisierende User gerade an einem Element arbeitet, dann wird logischer Weise der Heftinhalt erst nach Verlassen des Bearbeitungsmodus aktualisiert.

```
$scope.poll = function(){
    $timeout(function() {
        var content = $scope.notebook['content'];
        $http({
            method: 'POST',
            url: '/api/get_notebook',
            data: {id: $stateParams.id}
        }).success(function (data) {
            $scope.notebook = JSON.parse(data['notebook']);
            $scope.content = $scope.notebook['content'];
            if(JSON.stringify($scope.content) != JSON.stringify(content)) {
                $scope.update();
            }
            $scope.poll();
        });
```

```
    }, 10000);
};

$scope.poll();
```

Code 15 PWS Polling

12 Optical Character Recognition

Um Bilder hochladen zu können und eine Bild-zu-Text-Analyse zu ermöglichen wurde ein OCR-Modul implementiert. Die Idee dahinter ist, dass die Benutzer unserer Hefte möglichst einfach Bilder auswählen und einer Zeichenanalyse (OCR) unterziehen können. Dabei werden prinzipiell die Dateiformate *.jpeg .png* und *.gif* akzeptiert.

Bei der optischen Zeichenanalyse (OCR) wird der am Bild vorhandene Text durch eine OCR-Engine analysiert und in ein Textformat umgewandelt. Für unsere Lösung wurde das python-Framework *pytesseract* mit der OCR-Engine tesseract benutzt.

Um ein Bild zu analysieren klickt der User zunächst auf folgenden Button im Heft:

Abbildung 14 OCR-Button

Darauf öffnet sich ein Dialog mit einem Dateieingabefeld:

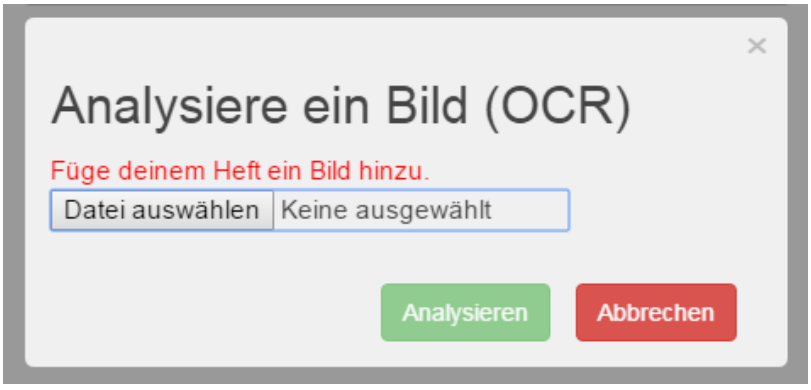

Abbildung 15 OCR-Dialog

Hier kann nun eine Bilddatei angegeben werden. Um die Funktionalität zu veranschaulichen wurde hier folgendes Beispielbild namens *testy.jpg* ausgewählt:

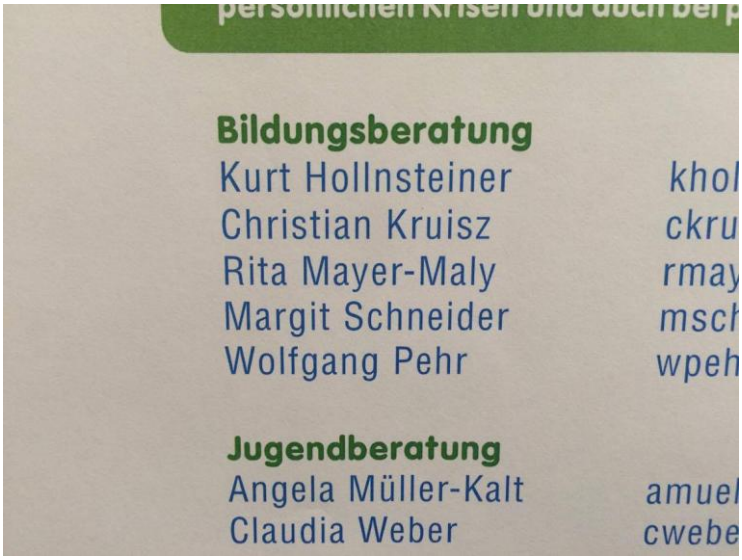

Abbildung 16 Beispielbild testy.jpg

Nachdem auf den Button "Analysieren" geklickt wurde wird das Bild auf den Server geladen und mittels OCR analysiert. Das Ergebnis dabei ist ein neu erstelltes Textelement im Heft mit dem textuellen Inhalt des angegebenen Bildes.

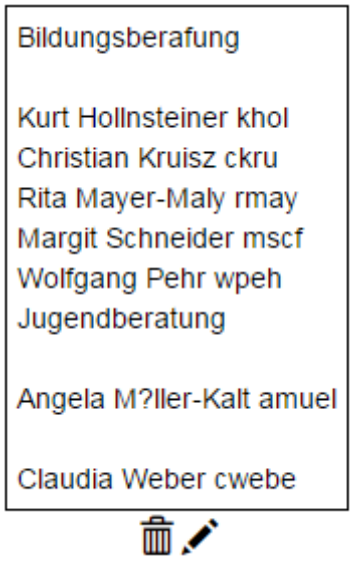

Abbildung 17 OCR generiertes Textelement

Wie im Textelement zu erkennen ist, funktioniert die OCR-Analyse mit der tesseract-Engine etwa zu 90%. Die restlichen 10% des Textes auf dem eingelesenen Bild wurde entweder falsch oder garnicht erkannt. Tests mit Bildern auf denen Handgeschriebenes zu sehen war schlugen allesamt fehl.

Wie schon erwähnt erscheint bei Klick auf den OCR-Button ein Dialog in welchem man die zu analysierende Bilddatei angeben kann. Bei Klick auf den Button "Analysieren" wird folgende Subroutine aufgerufen:

```
$scope.uploadOCRFile = function(){
        var file = $scope.ocrFile;
        if((file.type == "image/jpeg" || file.type == "image/png" ||
file.type == "image/gif") && file.size < 5242880) { //5MByte
```

```
            $scope.errormessage = "";
            var uploadUrl = "/api/analyseOCR";
            var message = fileUpload.uploadFileToUrl(file, uploadUrl);
            message.then(function(data) {
                data_data = "{\"data\":\""+data['ocrt']+"\"}";
                $scope.addelement('textarea', data_data);
                $window.location.reload();
                $window.location.reload();
            });
            ngDialog.close({
                template: 'ocrFileDialog',
                controller: 'notebookEditCtrl',
                className: 'ngdialog-theme-default',
                scope: $scope
            });
        }else{
            if(file.size < 5242880) {
                $scope.errormessage = "file size is more than 5MB";
            }else{
                $scope.errormessage = "filetyp is not supported";
            }
        }
    };
```

Code 16: UploadOCRFile-Methode

Hier ist zu sehen, dass zunächst der Dateityp und die Dateigröße überprüft wird. Wenn alles den Kriterien entspricht wird die Methode `uploadFileToUrl` aufgerufen. Unter der URL `url(r'^api/analyseOCR', 'dsn.views.notebook_views.view_analyseOCR', name="analyseOCR")` findet sich die Python-Methode `view_analyseOCR()`.

In dieser Methode erhält das File einen neu und zufällig generierten Namen und wird auf den Server lokal hochgeladen. Dann wird schließlich die Methode die für das eigentliche Analysieren zuständig ist aufgerufen:

```
ocrtext=analyseOCR(os.getcwd()+"/dsn/static/upload/"+filename+"."+typ)
```

Code 17: OCR-Analyse mit hochgeladenem File

In der Methode analyseOCR kommt das pytesseract-Framework zum Einsatz:

```
def analyseOCR(file):
    s =
    str(pytesseract.image_to_string(Image.open(file)).encode(sys.stdout.enc
    oding, errors='replace'))
    s = s[2:-1]
    s = s.replace("\\n", "<br />").replace("\\x", "")
    return s
```

Code 18: analyseOCR mittels pytesseract

Nach der Analyse wird der erkannte Text als JSONResponse zurückgegeben und das lokal hochgeladene Bild wird wieder vom Server gelöscht.

```
os.remove(os.getcwd()+"/dsn/static/upload/"+filename+"."+typ)
return JsonResponse({'ocrt': ocrtext})
```

Code 19: Bildentfernung und return des OCR-Textes

Abschließend wird mit dem daraus analysierten Text ein neues Textelement im Heft erzeugt. Dazu wird die addelement Methode aufgerufen.

13 Abbildungsverzeichnis

14 Codeverzeichnis

15 Quellen

[1] Christopher Thielen, “ui-router”, https://github.com/angular-ui/ui-router, zuletzt besucht: 30.10.2015

[2] Rahil Shaiksh, “Google reCaptcha with AngularJS”, http://code.ciphertrick.com/2015/05/19/google-recaptcha-with-angularjs/, zuletzt besucht: 30.10.2015

[3] Marijn Haverbeke, "Codemirror", http://codemirror.net/, zuletzt besucht: 25.11.2015

[4] The jQuery Foundation, „Draggable Widget“, http://api.jqueryui.com/draggable/, zuletzt besucht: 16.12.2015