Webservice - Frontend

Projekt: DigitalSchoolNotes

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

Version 2.3 **16.12.2015** **Status: [RELASE]**

	Datum	Name	Unterschrift
Erstellt	30.10.2015	Niklas Hohenwarter	
Geprüft			
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 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 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.

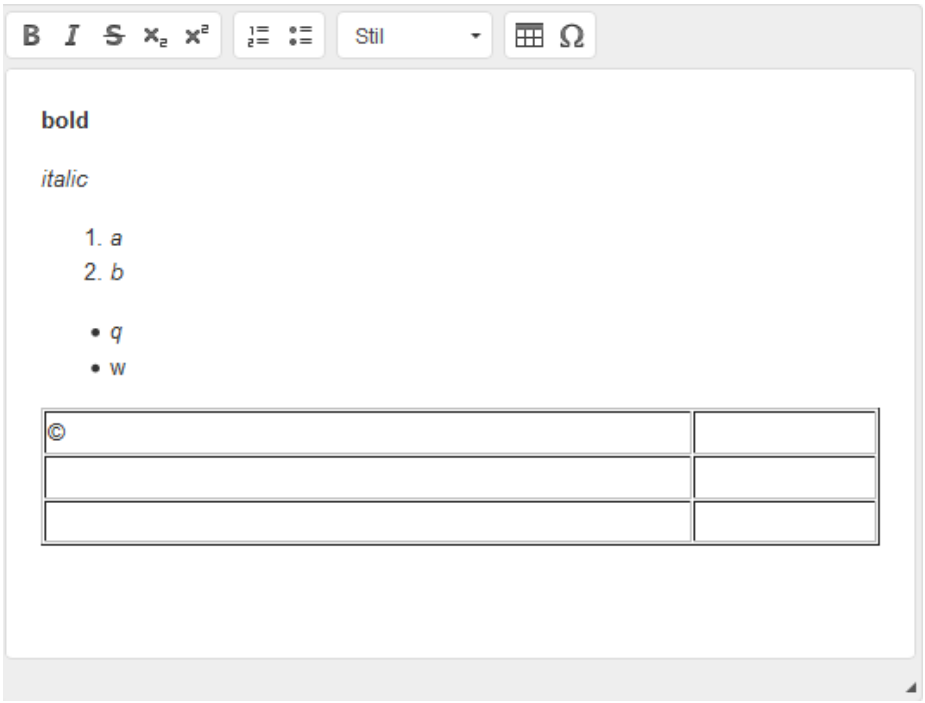

Abbildung 1: Texteditor

```
<main>
    <div class="adjoined-bottom">
        <div class="grid-container">
            <div class="grid-width-100">
                <div id="editor">
                    <h1>Text</h1>
                </div>
            </div>
        </div>
    </div>
</main>
<script>
    initSample();
</script>
```

Code 4: Einbinden des Texteditors

7 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)

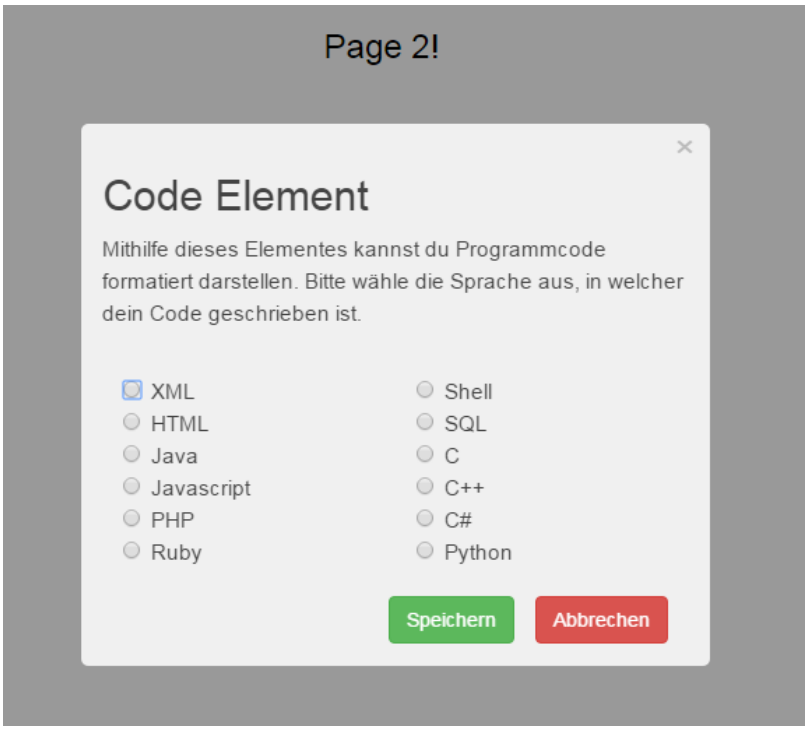

Abbildung 3 Programmiersprache auswählen

Es erscheint anschließend das (leere) Codelement im Heft.

Abbildung 4 Code Element

Unter dem Codeelement gibt es noch zwei 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).

Abbildung 5 Code Bearbeitung

In notebookedit.js sind zwei verschiedene Modes (readonly und schreibend) mit bestimmten Codemirror-Eigenschaften definiert.

```
// CODE ELEMENT
$scope.codeLanguage="text/xml";


$scope.cmOption = {
    lineNumbers: true,
    indentWithTabs: true,
    lineWrapping: true,
    scrollbarStyle: "null",
    onLoad: function (_cm) {
        $scope.modeChanged = function (id) {
            $scope.models['code'][id][1] = $scope.codeLanguage;
            _cm.setOption("mode", $scope.models['code'][id][1]);
        };
    }
};

$scope.ROcmOption = {
    lineNumbers: true,
    indentWithTabs: true,
    lineWrapping: true,
    scrollbarStyle: "null",
    readOnly: 'nocursor',
    onLoad: function (_cm) {
        $scope.modeChanged = function (id) {
            $scope.models['code'][id][1] = $scope.codeLanguage;
            _cm.setOption("mode", $scope.models['code'][id][1]);
        };
    }
};
```

Code 5 Codemirror Konfiguration

8 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 6: 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 7: 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.

9 Abbildungsverzeichnis

10 Codeverzeichnis

11 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