Webservice - Frontend

Projekt: DigitalSchoolNotes

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

Version 1.2 **25.11.2015** **Status: [RELASE]**

	Datum	Name	Unterschrift
Erstellt	30.10.2015	Niklas Hohenwarter	
Geprüft	25.11.2015	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 Routing

Um einzelne Zustände in unserer Applikation über eine URL aufrufen zu können, bentigen wir den ui-router[1]. Dieser ermöglicht es uns, aufgrund einer URL einen Controler zuzuweisen. Desweiteren basiert routing nicht mehr nur auf URLs sondern auf states. Das bedeutet, dass ich durch z.B. einen Klick auf einen Button in einen anderen Status weitergeleitet werde. 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?
        }
    });

});
```

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) {

            });
    }
}
```

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

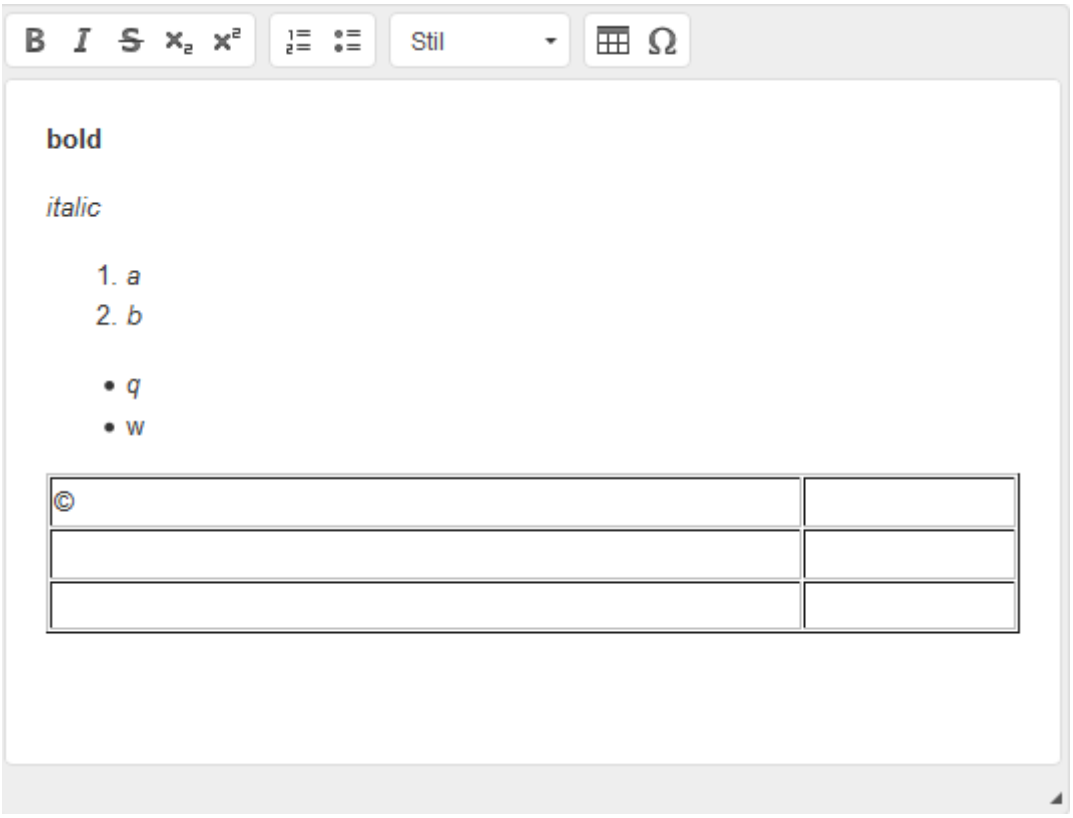
Falls eine Menge von Daten angezeigt werden müssen, ist es besser diese über mehrere Seiten zu verteilen. Das wird Pagination genannt. Für die Realisierung benötigt man die Anzahl der Objekte die insgesamt ausgegeben werden sollen. Dann definiert man wieviele Elemente pro Seite angezeigt werden sollen. Um die Seitenanzahl zu berechnen, dividiert man alle Elemente durch die Anzahl der Elemente, 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;
};
```

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.




```
<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>
```

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
- JavaScript
- C-ähnliche Sprachen
- Python

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

Der Code-Editor erscheint nun auf der Seite und der User kann in einer Dropdown-Liste eine Sprachen aussuchen. Hier sieht man eine Beispielabbildung des Code-Elements:

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <xs:element name="Schule">
3   <xs:complexType>
4   <xs:sequence>
5   <xs:element name="Abteilung">
6   <xs:complexType>
7   <xs:sequence>
8   <xs:element name="Klasse" maxOccurs="30" minOccurs="0'
9   <xs:complexType>
10  <xs:sequence>
11  <xs:element type="xs:string" name="Schueler" maxOccurs="100"
12  </xs:sequence>
13  <xs:attribute type="xs:string" name="kname"/>
14  </xs:complexType>
15
```

Mode : XML

In *notebook.html* steht dann der folgende Tag:

```
<div compile="divHtmlVar" ></div>
```

In notebookedit.js habe ich folgende function angelegt, die aufgerufen wird wenn der Code-Button betätigt wird:

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

      $scope.divHtmlVar = $scope.divHtmlVar + '<section> <textarea
      class="codestyle" rows="6" cols="70" ui-
      codemirror="cmOption"></textarea> Mode : <select ng-model="mode" ng-
      options="m for m in modes" ng-change="modeChanged()"></select>
      </section>';

}
```

Ich beschreibe die verfügbaren Modes:

```
    $scope.modes = ['Scheme', 'XML', 'Javascript', 'clike', 'python'];

    $scope.mode = $scope.modes[0];
```

Konfiguration der zusätzlichen Optionen für Codemirror:

```
$scope.cmOption = {

        lineNumbers: true,

        indentWithTabs: true,

        onLoad: function (_cm) {

                    $scope.modeChanged = function () {

              _cm.setOption("mode", $scope.mode.toLowerCase());

          };

      }

};
```

8 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