**Dokumentation Webprogrammieren**

****

**Mitgliederverwaltung**

Webprogrammieren Studienarbeit | 4. Semester

Dozent: Ulrich Hauser

Hochschule für Technik und Wirtschaft

Bachelorstudiengang: Digital Business Management, Teilzeit

Chur, 8. Mai 2019

Inhaltsverzeichnis

**Geben Sie die Kapitelüberschrift ein (Stufe 1)1**

Geben Sie die Kapitelüberschrift ein (Stufe 2)2

Geben Sie die Kapitelüberschrift ein (Stufe 3)3

**Geben Sie die Kapitelüberschrift ein (Stufe 1)4**

Geben Sie die Kapitelüberschrift ein (Stufe 2)5

Geben Sie die Kapitelüberschrift ein (Stufe 3)6

**Auftragsskizze**Übersicht

Es geht im Projekt darum, Mitglieder eines Sportvereins zu verwalten. Mitglieder sollen unter anderem hinzugefügt, geändert, gelöscht werden können. Mitglieder sollen ausserdem mit bestimmten Rechten auch andere Mitglieder ansehen bzw. Bearbeiten können.

Mitgliederdaten

Mitglieder verfügen über:

•Adresse (Strasse, Nummer, PLZ, Wohnort)

•Geburtsdatum

•Eintrittsdatum

•Aktivitätsstatus (aktiv/passiv/ausgetreten etc.)

•Klassifizierung (Senior/Seniorin, Erwachsene Person, Jugendlich, Kind)

•eine Graduierung (im Turnen z.B. K1, K2, K3, im Kampfsport 3. Kyu oder 2. Dan etc.)

•Beziehungen zwischen Mitgliedern wie ‘verheiratet mit’, ‘Kind von’ etc.

•… (im Laufe des Projektes entwickelt sich hier evtl. noch mehr, bzw andere Anforderungen werden fallengelassen

Form

Die Verwaltung soll über eine Web-Page erreichbar sein. Die Datenbank soll über SQL-Abfragen angesprochen werden können (MariaDB, eine open source mySql-Variante bevorzugt, aber nicht verpflichtend).

Die genaueren Anforderungen wachsen im Backlog über das Semester weg.

**Kickoff & Organisation**

In einem Kickoff Meeting konnten die grundlegenden organisatorischen Belangen geregelt werden. Dazu wurde das Projekt kurz visualisiert und auf die kleinstmöglichen User Stories runtergebrochen, um anschliessend die Arbeitszuteilung vorzunehmen und eine bessere Übersicht schaffen zu können. Im ersten Moment wählte man einen Projektleiter, welcher genügend Background aus dem IT Bereich mitbringt, um Gruppenübergreifend koordinieren und unterstützen zu können, siehe Protokoll. Darauf hin einigte man sich auf die Unterteilung in zwei Gruppen und zwei jeweilige Mentoren, mit dem einen Hauptfokus der Anzeige der Mitglieder inklusive der Gesamtübersicht, sowie der zweiten Gruppe mit dem Fokus auf die Erfassung der Mitglieder inklusive Datenbankschnittstelle. Als weiterer Task des Kickoff Meetings wurden die zu verwendenden Kanäle für die Kommunikation und den Datenaustausch definiert und zugleich kurz erklärt, dazu zählen Slack, KanBoard, Github / GitCola, GoogleDrive und für Ausnahmefälle eine Whatsappgruppe.

**View 2 | HTML Rohkonstrukt Mitgliedererfassung**

Für die Erstellung der Hauptfelder zur Eingabe der Stammdaten bei der Benutzererfassung wurde ein Branch vom Master angelegt mit dem Namen: «MemberFields»

Im Branch Memberfields wurde als erstes das rohe HTML Konstrukt angelegt zur Erstellung des grundsätzlichen Layouts für die Mitglied- / Membererfassung in Form eines Anmeldeformulars:

Formularbeginn

Formularende

Formularbeginn

Formularende

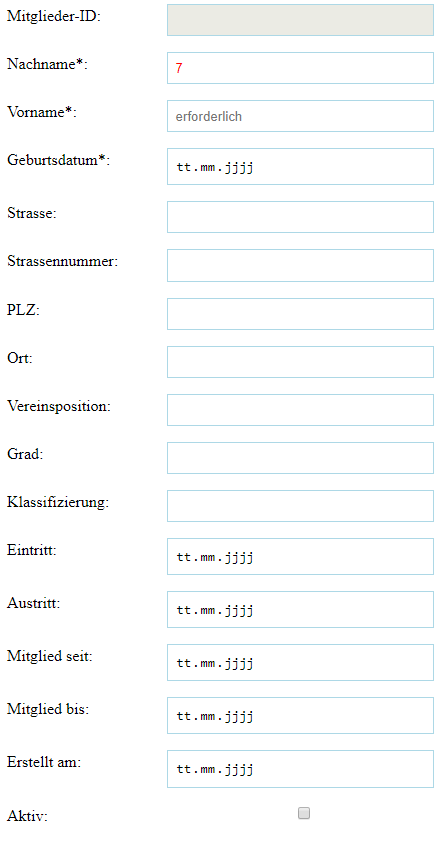
|  |  |
| --- | --- |
|  | <html xmlns="https://www.w3.org/1999/xhtml"> |
|  | <head> |
|  | <title>Mitgliederverwaltung</title> |
|  | <link rel="stylesheet" type="text/css" href="..\style\members.css"> |
|  | <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script> |
|  | <script src="../js/members.js"></script> |
|  | </head> |
|  | <body> |
|  | <!-- Mitglieder-ID, Kein Input möglich--> |
|  | <p>Mitglieder-ID:</p> |
|  | <input type="text" name="m\_id" disabled><br> |
|  | <!-- Inputfelder--> |
|  | <p>Nachname\*:</p> |
|  | <input type="text" name="nachname"><br> |
|  | <p>Vorname\*:</p> |
|  | <input type="text" name="vorname"><br> |
|  | <p>Geburtsdatum\*:</p> |
|  | <input type="date" name="geburtsdatum"><br> |
|  | <p>Strasse:</p> |
|  | <input type="text" name="strasse"><br> |
|  | <p>Strassennummer:</p> |
|  | <input type="text" name="strnummer"><br> |
|  | <p>PLZ:</p> |
|  | <input type="text" name="plz"><br> |
|  | <p>Ort:</p> |
|  | <input type="text" name="ort"><br> |
|  | <p>Vereinsposition:</p> |
|  | <input type="text" name="vereinsposition"><br> |
|  | <p>Grad:</p> |
|  | <input type="text" name="grad"><br> |
|  | <p>Klassifizierung:</p> |
|  | <input type="text" name="klassifizierung"><br> |
|  | <p>Eintritt:</p> |
|  | <input type="date" name="eintritt"><br> |
|  | <p>Austritt:</p> |
|  | <input type="date" name="austritt"><br> |
|  | <p>Mitglied seit:</p> |
|  | <input type="date" name="von\_dat"><br> |
|  | <p>Mitglied bis:</p> |
|  | <input type="date" name="bis\_dat"><br> |
|  | <p>Erstellt am:</p> |
|  | <input type="date" name="erstellt"><br> |
|  | <p>Aktiv:</p> |
|  | <input type="checkbox" name="aktiv"><br> |
|  | <!-- neuer Umbruch --> |
|  | <br> |
|  | <!-- Buttons--> |
|  | <button type="button" class="btn">Erstellen</button> |
|  | <button type="button" class="btn">Ändern</button> |
|  | <button type="button" class="btn">Löschen</button> |
|  | </body> |
|  | </html> |
|  |  |

Die einzelnen Eingabefelder werden wie ersichtlich mit «inputs» definiert im Body. Als nächstes werden mittels dem eingebundenen JavaScript File die Validierungen angelegt. Die Validierungen dienen dazu, die Daten in den Eingabefelder gemäss den Inkrementvorgaben zu überprüfen, sodass der Benutzer keine falschen Datentypen in die Felder einfügen kann. Sprich für die Postleitzahl werden rein Zahlen zugelassen und keine Buchstaben etc.

**View 2 | Erste Formularinteraktion mit Farbwechsel nach Benutzereingabe**

|  |  |
| --- | --- |
|  | /\* |
|  | initialisation of the dynamic part of the web site |
|  | \*/ |
|  | $(document).ready(init); |
|  |  |
|  | /\*Beim Klicken auf Feld, Farbe ändern - INIT Funktion an erster Stelle\*/ |
|  | function init(){ |
|  | $("input") |
|  | .focus(enterFieldColor) |
|  | .blur(leaveFieldColor); |
|  | } |
|  | /\*Farbe von Feld ändert sich in grau wenn man in Feld klickt\*/ |
|  | function enterFieldColor(){ |
|  | $(this) |
|  | .css("backgroundColor", "silver") |
|  | } |
|  |  |
|  | /\*Farbe von Feld ändert sich in weiss wenn man Feld verlässt\*/ |
|  | function leaveFieldColor(){ |
|  | $(this) |
|  | .css("backgroundColor", "white") |
|  | } |
|  |  |

Wie im Code ersichtlich wurde eine Style Funktion angelegt via JavaScript, sodass beim Klicken in ein Feld die Hintergrundfarbe auf «Silver» ändert. Dies dient der Benutzerfreundlichkeit und soll dem User bei der Führung durch das Dokument behilflich sein.

****

**View 2 | HTML-Validierung Formular**

Als nächstes wurde für die Benutzer ID als erstes Feld der Eingabe-Maske eine Funktion in JavaScript angelegt, welche das Feld als Autoincrement hinterlegt und es verunmöglicht, in das Feld zu schreiben. Die Mitglieder ID wird automatisch zugeordnet und soll nicht frei wählbar sein. Im html File kann dies durch den Zusatz im Input mit *disabled* vergewissert werden, sodass die Eingabe für den Benutzer nicht möglich ist.

Weiter wurde das Formular in den offiziellen Tag *<form>* gepackt und um die jeweiligen Inputs erweitert, damit die Verknüpfung mit den Javascript Funktionen hergestellt werden kann und um weitere Einschränkungen zu definieren. Jedem Input, also jedem Datenfeld, wurde eine *id* verliehen für eine saubere Selektion. Mit *maxlength* wurde die maximale Zeichenlänge hinterlegt, mit *placeholder* können Datenfelder vorausgefüllt werden als Hilfestellung für den Benutzer, durch *required* wird die Eingabe in einem Feld erforderlich vor dem Absenden und mit *onchange* werden die Validierungsfunktionen durch Javascript verknüpft.

<form>

<!-- Mitglieder-ID, Kein Input möglich-->

<p><label>Mitglieder-ID:</label>

<input id="m\_id" type="text" name="m\_id" disabled></p>

<!-- Inputfelder-->

<p><label>Nachname\*:</label>

<input id="nachname" type="text" name="nachname" onchange="nachNaValidation()" maxlength="30" placeholder="erforderlich" required></p>

<p><label>Vorname\*:</label>

<input id="vorname"type="text" name="vorname" onchange="vorNaValidation()" maxlength="30" placeholder="erforderlich" required></p>

<p><label>Geburtsdatum\*:</label>

<input id="geburtsdatum"type="date" name="geburtsdatum" required></p>

<p><label>Strasse:</label>

<input id="strasse"type="text" name="strasse" onchange="strasseValidation()" maxlength="30"></p>

<p><label>Strassennummer:</label>

<input id="strassennummer"type="text" maxlength="4" name="strnummer"></p>

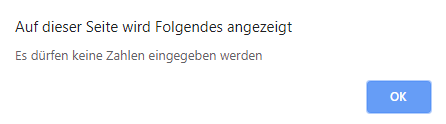
<p><label>PLZ:</label>

<input id="plz" type="text" name="plz" onchange="plzValidation()"></p>



**View 2 | Javascript-Validierung**

Die Funktionen für die Eingabevalidierung bei der Mitgliedererfassung wurden mit Javascript und Regex umgesetzt. Letzteres der Einfachheit halber und um grosse for- und while- Schleifen zu vermeiden. Konkret wird eine Variable erstellt welche die Regex Vorgaben erfüllt und eine zweite, welche die Benutzereingabe mit der Regex Variable abgleicht. Insofern die Eingaben nicht übereinstimmen, wird ein Alert ausgegeben und die Schriftfarbe in rot gewechselt. Die Funktionen unterscheiden sich anschliessend nur noch durch die Regex Vorgabe und den jeweiligen Alert, der Rest bleibt gleich.



// Validierung beim Nachname

function nachNaValidation(){

var regEx = "^[a-zA-Z]+$";

var nachNaEingabe = document.getElementById("nachname");

if (!nachNaEingabe.value.match(regEx)){

nachNaEingabe.style.color ="red";

alert("Es dürfen keine Zahlen eingegeben werden");

} else {

nachNaEingabe.style.color ="black";

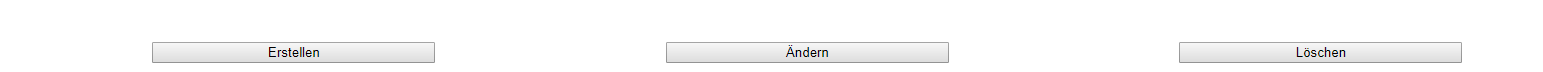
}  
}

**View 2 | HTML-Buttons**

Der erste Button zum Absenden des Erfassungsformulars wird erstellt mit dem *<button>* Tag. *Type submit* und *value Absenden* als Indikator für die Funktion des Knopfes. Mit *onclick* wird die Javascript Funktion verknüpft, mit welcher anschliessend die eingegebenen Daten in der Consol gespeichert und strukturiert werden, um sie in der Datenbank zu implementieren. Zudem wurde eine Klasse erstellt für die weiteren Buttons auf der Seite.

<!-- Buttons -->

<button id="buttonCreate" type="submit" value="Absenden" class="btn" onclick="createMember()">Erstellen</button>



**View 2 | Javascript-Buttons**

In der Javascript Funktion für die Buttons werden sämtliche Benutzereingaben in einer Variablen in Form eines *Objects* gespeichert, welche anschliessend wieder in der Consol.log ausgegeben werden können.

function createMember(){

var member = new Object();

member.nachname = document.querySelector("#nachname").value;

member.vorname = document.querySelector("#vorname").value;

member.geburtsdatum = document.querySelector("#geburtsdatum").value;

member.strasse = document.querySelector("#strasse").value;

member.strassennummer = document.querySelector("#strassennummer").value;

member.plz = document.querySelector("#plz").value;

member. ort = document.querySelector("#ort").value;

member.vereinsposition = document.querySelector("#vereinsposition").value;

member.grad = document.querySelector("#grad").value;

member.klassifizierung = document.querySelector("#klassifizierung").value;

member.eintritt = document.querySelector("#eintritt").value;

member.austritt = document.querySelector("#austritt").value;

member.von\_dat = document.querySelector("#von\_dat").value;

member.bis\_dat = document.querySelector("#bis\_dat").value;

member.erstellt = document.querySelector("#erstellt").value;

member.aktiv = document.querySelector("#aktiv").value;

console.log(member);

console.log(member.vorname);

}