Validierung

„Never trust a user input“

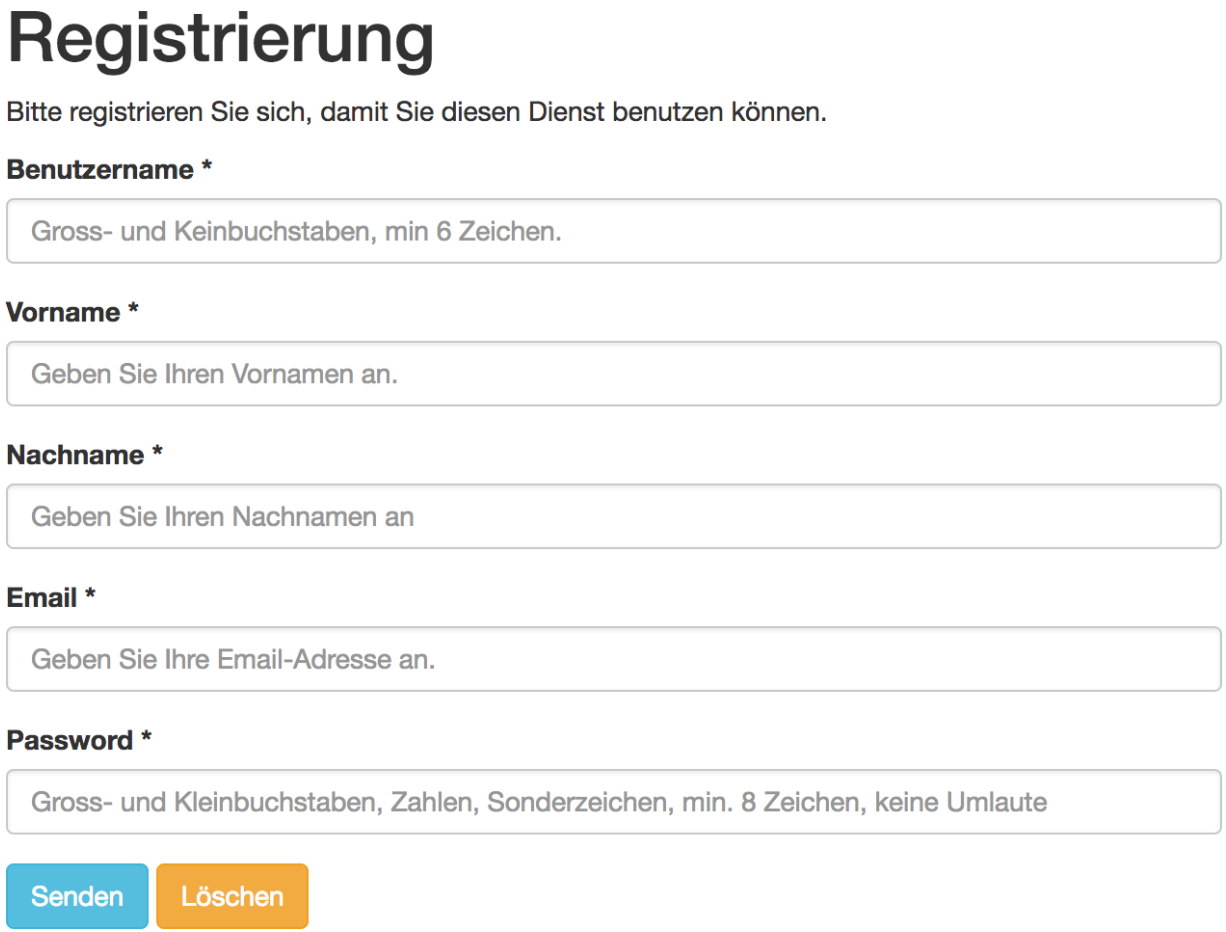
Clientseitige Validierung

	#	Name	Typ	Kollation	Attribute	Null
☐	1	**id**	bigint(20)		UNSIGNED	Nein
☐	2	**username**	varchar(30)	utf8_unicode_ci		Nein
☐	3	**fitstname**	varchar(30)	utf8_unicode_ci		Nein
☐	4	**lastname**	varchar(30)	utf8_unicode_ci		Nein
☐	5	**email**	varchar(100)	utf8_unicode_ci		Nein
☐	6	**password**	varchar(255)	utf8_unicode_ci		Nein

Als Ausgangslage dient meistens die Definition der Datenbank-Tabelle.

Wir können daraus ablesen, wie viele Zeichen der Benutzer maximal eingeben darf, von welchem Type das Eingabefeld sein soll und ob es sich um eine Pflichteingabe handelt.

Clientseitige Validierung



Daraus entsteht das HTML-Formular, in welchem der Benutzer seine Daten eingeben kann.

Die clientseitige Validierung unterstützt die **Usability** einer Webseite. Wir können den Benutzer auf das korrekte Format der Daten hinweisen, bevor er das Formular gesendet hat.

Clientseitige Validierung kann einfach umgangen werden, wenn Javascript ausgeschalten wird, oder ein nicht HTML5 konformer Browser verwendet wird.

Clientseitige Validierung – Input Typen

Einige der HTML5 Input-Typen prüfen die Benutzereingaben vor dem Senden auf Korrektheit.

```
<input type=„email“>
```
Prüft ob die Benutzereingabe ein @ enthält.

```
<input type=„number“>
```
Prüft ob die Benutzereingabe eine Zahl ist.

Weitere HTML5 Input-Typen finden Sie auf:
https://www.w3schools.com/html/html_form_input_types.asp

Clientseitige Validierung - Attribute

Das HTML-Element `<input>` bietet einige Attribute, welche die clientseitige Validierung unterstützen:

`required:`	macht ein Eingabefeld zu einem Pflichtfeld
`maxlength:`	beschränkt die maximale Eingabelänge
`max / min:`	beschränkt die Eingabe von Zahlen und Daten auf einen bestimmten Bereich
`pattern:`	ermöglicht die Eingabe gegen eine RegularExpression zu testen

Weitere HTML5 Input-Attribute finden Sie auf:
https://www.w3schools.com/html/html_form_attributes.asp

Serverseitige Validierung

Da wir nie wissen, ob die Benutzereingaben auf der Clientseite geprüft wurden, müssen alle Daten auf der Serverseite nochmals geprüft werden.

Formulardaten werden in der Regel mit der Methode POST gesendet.

Die `action` bestimmt, wohin die Daten gesendet werden.

```
<form action=„index.php“ method=„post“>
```

Anschliessend stehen die Daten im superglobalen, assoziativen Array `$_POST` zur Verfügung.

https://www.w3schools.com/php/php_forms.asp

Serverseitige Validierung

Der Wert des Eingabefeldes mit dem Attribut `name=„username“` steht in der Variable `$_POST[‚username‘]` zur Verfügung.

	#	Name	Typ	Kollation	Attribute	Null
☐	1	**username**	varchar(30)	utf8_unicode_ci		Nein

Wir prüfen nun auch Serverseitig, gemäss der Definition der Datenbank-Tabelle, ob der Wert 30 Zeichen nicht überschreitet `varchar(30)` und ohne Leerzeichen am Anfang und Ende mindestens ein Zeichen beinhaltet `Not Null.`

Serverseitige Validierung

Zusätzlich prüfen wir, um Fehlermeldungen zu verhindern, ob die Variable `$_POST['username']` vorhanden und nicht Null ist.

```
if(isset($_POST['username'])
        && !empty(trim($_POST['username']))
        && strlen(trim($_POST['username']) <=30)){
            $username = htmlspecialchars(trim($_POST['username']));
}
```

`isset()`:	prüft ob die Variable vorhanden und nicht NULL ist
`empty()`:	prüft ob eine Variable einen Wert enthält
`strlen()`:	prüft die Länge eines Stringes
`trim()`:	schneidet Leerzeichen am Anfang und Ende eins Stringes weg.
`htmlspecialchars()`:	Escaped Sonderzeichen in einem String und verhindert so Script-Injection.

Serverseitige Validierung

Weitere Möglichkeiten, eine Benutzereingabe serverseitig zu prüfen:

`preg_match()`: ermöglicht die Eingabe gegen eine RegularExpression zu testen.

`filter_var()`: Filtern einer Variablen mit einem angegebenen Filter z.B. Email-Adresse.

https://www.w3schools.com/php/php_form_url_email.asp

Anwenden

Lösen Sie nun die Aufgabe im Ordner:
BSCW > 133 > 2_1_Auftrag_Validierung.