Angular Formulare

Inhalt:

1)app.module.ts vorbereiten
2)neue Komponente anlegen und HTML-Formular (Bootstrap) anlegen
3)die neue Komponente mit Angular Tools verbessern – ngForm, ngModel, name
4)Validierung (Überprüfung) von Feldern
5)Anzeigen von Fehlermeldungen
6)Vor dem Absenden prüfen, ob die Form insgesamt passt
7)Daten an ein Model binden, um es an einen Server senden zu können

Formulare dienen zur strukturierten Verarbeitung von Benutzereingaben und sind ein wichtige Baustein für die Gewinnung und Auswertung von Daten.

Ein typisches Formular ist z.B. eines für die Neu-Registrierung von Kunden.

Man kann Formulare auch mit einfachem HTML erstellen, aber Angular-Code bietet viele Vorteile. So gibt es eine komfortable API, die mit einem in HTML definierten Formular zusammenarbeitet und mit der die Eingabedaten zentral ausgewertet und verarbeitet werden können. Somit werden Wert- und Zustandsänderungen überwacht und auf Änderungen wird reagiert. Z.B. kann das Absenden des Formulars automatisch blockiert werden, wenn ungültige Eingaben getätigt wurden. Diese Forms-API ist in dem Modul @angular/forms enthalten und kann ganz bequem über einen üblichen Import in das Projekt geholt werden (siehe dazu später mehr):

```
import { FormsModule } from '@angular/forms';
```

Grundsätzlich haben Formulare folgende Anforderungen zu bewältigen:

- Unterstützen der Eingaben der Benutzer
- Validierung (auf Fehler hinweisen und Absenden zurückhalten, bis Korrektur)
- Überwachung von Formularzuständen zur weiteren Steuerung
- Es soll nur möglich sein, ein Formular abzusenden, wenn es keinen Validierungsfehler mehr besitzt
- Validierungsfehler sollen nur sichtbar sein, wenn der Nutzer bereits einmal mit dem Fokus (Maus) innerhalb des Feldes war
- Nutzereingaben sollen sich sofort auf die Darstellung des Validierungsfehlers auswirken, nicht erst nach dem Verlassen des Feldes

Prinzipiell gibt es 2 Arten um Formulare in Angular zu erstellen.

1. template driven forms (TDF)
2. reactive forms

Wir verwenden hier die TDF, also die Erstellung im Template und somit in der HTML-Datei der Componente.

Die Reactive Form-Variante orientiert sich eher am Code in der TypeScript-Datei sowie der Komponentenklasse und eignet sich vor allem für ziemlich komplexe Eingabeformulare. Dabei wird die Formularlogik komplett in TypeScript bzw. JavaScript implementiert.

Template Driven Forms

Template driven forms bieten eine einfache Möglichkeit, Formulare mit Angular zu initialisieren und auszuwerten. Angular bindet sich dabei automatisch an ein <form>-Element und liest die Daten der Eingabefelder, Radio-Buttons und Checkboxen aus. Die Formulareingaben werden automatisch in einem Objekt gespeichert.

Zusätzlich kümmert sich Angular automatisch um Validierung und die Anzeige von Meldungen.

Hat man, so wie wir, das Projekt mit der Angular CLI erstellt, hat das Tool schon automatisch einiges vorbereitet.

1)app.module.ts vorbereiten

Daher muss man nur noch in der Datei „app.module.ts“ folgendes importieren:

import { FormsModule } from '@angular/forms';

```
6   import { ImpfungComponent } from './impfung/impfung.component
7   import { FormsModule } from '@angular/forms';
8
```

und in den „imports“ bekannt machen:

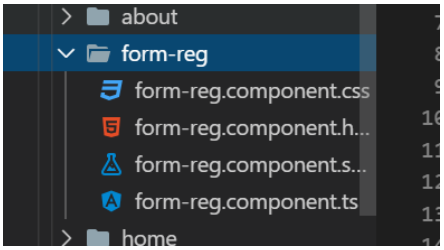
```
16  imports: [
17    BrowserModule,
18    AppRoutingModule,
19    HttpClientModule,
20    FormsModule
21  ],
22  providers: [],
```

2)neue Komponente anlegen und HTML-Formular anlegen (Bootstrap)

Erstelle eine neue Komponente, in der das Formular erstellt werden soll.

ng g c form-reg

Daher ergibt sich folgendes:



Damit es anklickbar ist, müssen 2 Schritte erledigt werden:

- die Route anlegen in „app.routing.ts“
- in der app.component.html in die Navigation einfüge

```
app.module.ts   app-routing.module.ts
src > app > app-routing.module.ts > ...
24    { path: 'schueler', component: SchuelerComponent},
25    { path: 'impfung', component: ImpfungComponent},
26    { path: 'form-reg', component: FormRegComponent}
27  ];
```

Inkl. automatischem import:

```
app.module.ts   app-routing.module.ts
src > app > app-routing.module.ts > ...
1   import { FormRegComponent } from './form-reg/form-reg.component';
2   import { NgModule } from '@angular/core';
```

app.component.html:

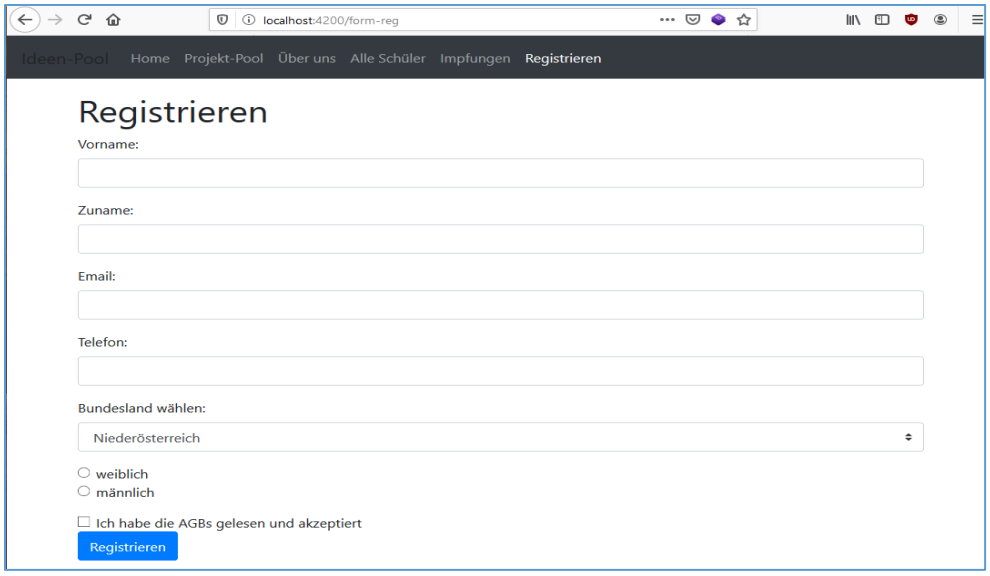
```
app.module.ts   app.component.html   app-routing.module.ts
src > app > app.component.html > div.container > nav.navbar.navbar-expand-lg.navbar-dark.bg-dark.fixed-top > div#navbarNav.co
17        <li class="nav-item">
18        <a routerLink='/impfung' routerLinkActive="active" class="nav-link">Impfungen</a>  </li>
19        <li class="nav-item">
20          <a routerLink='/form-reg' routerLinkActive="active" class="nav-link">Registrieren</a>  </li>
21    </ul>
```

Öffne die „html“ und schreibe ein einfaches HTML-Formular, das erst im Nachhinein auf die Angular typische Formulierung gebracht werden wird.

Dabei werden folgende Elemente verwendet:

- input
- Auswahlmöglichkeiten (select)
- Radiobutton – beachte, dass sie den gleichen Namen haben; den Wert bestimmt hier das value-Attribut
- Checkbox
- submit

Das Ergebnis wird so aussehen:



Code:

```
<br><br><br>
 <h1>Registrieren</h1>
  <form>
   <div class="form-group">
    <label>Vorname:</label>
    <input type="text" name="vorname" class="form-control">
   </div>

   <div class="form-group">
    <label>Zuname:</label>
    <input type="text" name="zuname" class="form-control">
   </div>

   <div class="form-group">
    <label>Email:</label>
    <input type="email" name="email" class="form-control">
   </div>

   <div class="form-group">
    <label>Telefon:</label>
    <input type="tel" name="telefon" class="form-control">
   </div>

   <div class="form-group">
    <label>Bundesland wählen:</label>
    <select class="custom-select">
       <option selected>Niederösterreich</option>
       <option>Oberösterreich</option>
       <option>Salzburg</option>
       <option>Steiermark</option>
       <option>Klagenfurt</option>
       <option>Tirol</option>
       <option>Kärnten</option>
       <option>Wien</option>
       <option>Burgenland</option>
     </select>
   </div>

   <div class="mb-3">
    <div class="form-check">
      <input class="form-check-input" type="radio" name="geschlecht" value="weiblich">
      <label class="form-check-label">weiblich</label>
    </div>
    <div class="form-check">
      <input class="form-check-input" type="radio" name="geschlecht" value="männlich">
      <label class="form-check-label">männlich</label>
    </div>
   </div>

   <div class="form-check">
      <input class="form-check-input" type="checkbox">
      <label class="form-check-label">Ich habe die AGBs gelesen und akzeptiert</label>
```

```
  </div>

   <button class="btn btn-primary" type="submit">Registrieren</button>
</form>
```

3)die neue Komponente mit Angular Tools verbessern

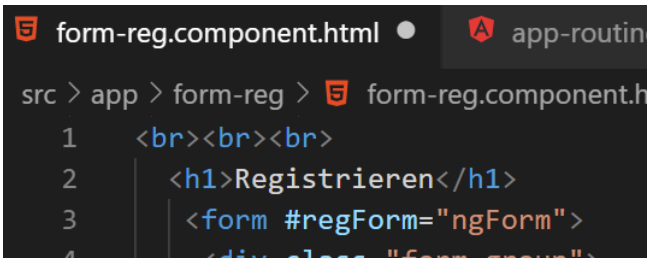
Da das FormModule in „app.module.ts“ bereits importiert ist, wirken sich die nächsten Änderungen in Richtung Angular auch wirklich aus.

https://www.youtube.com/watch?v=iyabqUWYsz4&list=PLC3y8-rFHvwhwL-XH04cHOpJnkgRKykFi&index=5

Nun verwendet man die ngForm- und ngModel-Direktives.

a) ngForm

Als erstes füge der <form> in Zeile 3 die Direktive „ngForm“ hinzu. Der Name für die Variable kann frei gewählt werden. Davor wird die Raute gesetzt: hier wurde der Name „#regForm“ gewählt, da es sich um ein Registrierungsformular handelt und weitere folgen könnten, die dann anders heißen.



Alleine der Import des FormsModules hat dafür gesorgt, dass alle Formulare der Applikation automatisch eine Instanz der Direktive „NgForm“ erzeugen. Sie wird somit auf alle Formulare angewendet und exportiert sich selbst über den Bezeichner „ngForm“. Durch unsere Zuweisung von #regForm=“ngForm“ erhält man somit Zugriff auf die gesamte Schnittstelle der Direktive.

b)ngModel

Jedes Form-Element (input, select, radio…) erhält eine Direktive „ngModel“. Dies dient zur weiteren Auswertung und Verarbeitung der Daten. Diese geht immer gemeinsam mit dem „name-Attribut“ von Formularfeldern einher. Die Direktive legt sogenannte Form-Controls an und bindet diese an das name-Attribut. Ein Form-Control hat Zugriff auf den Wert des Eingabefeldes. Außerdem enthält es Informationen zur Validität und zum Status (Wurde das Feld berührt? Wurde ein Wert geändert? Etc.).

Füge überall ein „ngModel“ ein:

```
<div class="form-group">
  <label>Zuname:</label>
  <input type="text" name="zuname" class="form-control" ngModel>
</div>
```

Bis hin zu

```
<div class="form-check">
    <input class="form-check-input" type="checkbox" ngModel>
    <label class="form-check-label">Ich habe die AGBs gelesen und akze
</div>
```

Hier hat bereits jedes Element ein name-Attribut. Das ist auch wichtig, ohne dem geht es nicht!

c) name-Attribut – für jedes Feld

Da aber noch einige fehlen müssen diese nun auch ein name-Attribut erhalten:
fehlt bei: Bundesland und der Checkbox bei den AGBs.

```
<div class="form-group">
  <label>Bundesland wählen:</label>
  <select class="custom-select" ngModel>
    <option selected>Niederösterreich</option>
```

Füge ein: name="bundesland"

```
<label>Bundesland wählen:</label>
<select class="custom-select" ngModel name="bundesland">
  <option selected>Niederösterreich</option>
```

Und bei der Checkbox „agb“

```
<div class="form-check">
  <input class="form-check-input" name="agb" type="checkbox" ngModel>
  <label class="form-check-label">Ich habe die AGBs gelesen und akzepti
```

Info:
Bestimmte Elemente stammen aber nicht aus dem FormModule von Angular, sondern sind normale Elemente von Bootstrap. So z.B. die CSS-Klasse „form-control“ oder „form-group“ oder „btn“. Sie sind für das Design zuständig und haben keine Auswirkung auf die Funktionsweise des Formulars.

d)Testen

Das Formular funktioniert schon einwandfrei. Füge dazu zum Testen eine Interpolation mit den doppelten geschwungenen Klammern ein, kann man die Einträge direkt beobachten:

Den Namen der Form (ohne der Raute) gefolgt von „value“, dem geraden Strich (bei der Taste <) und dann „json“

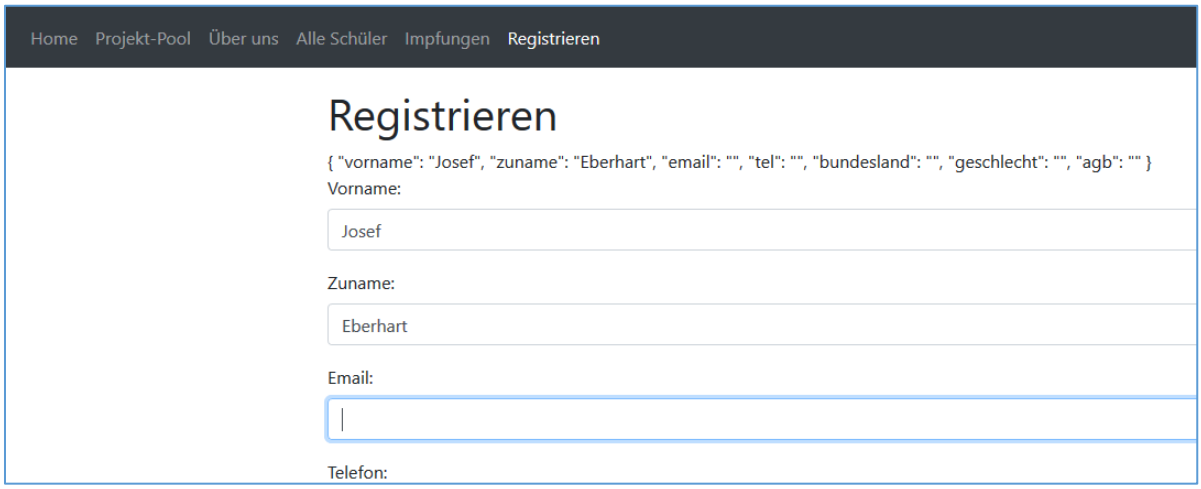
{{ regForm.value | json}}

```
<br><br><br>
  <h1>Registrieren</h1>
  <form #regForm="ngForm">

   {{ regForm.value | json}}

   <div class="form-group">
```

Als Ergebnis sieht man im Formular die Anzeige der eingegeben Daten. Gibt man etwas ein, wird es sofort angezeigt.



Alle Daten sind Properties des Objekts.

Entferne wieder diese Zeile:

```
{{ regForm.value | json}}
```

Info: Mit „regForm.value“ kann man später auch Daten an den Server senden, die aus dem Formular kommen.

4)Validierung (Überprüfung) von Feldern

Engl.: Field-Validation.

Je nach Zustand der Kontrollelemente werden automatisch vordefinierte CSS-Klassen für ein jeweiliges Eingabeelement gesetzt.

Verfügbare CSS-Klassen:

CSS-Klasse (if true)	CSS-Klasse (if false)	Beschreibung
ng-touched	ng-untouched	Der Wert wurde verwendet / nicht verwendet
ng-valid	ng-invalid	Der Wert ist gültig / ungültig
ng-dirty	ng-pristine	Der Wert wurde geändert / ist unberührt

Somit kann man diese CSS-Klasse nutzen, um ein

- Eingabefeld z.B. rot hervorzuheben, wenn kein Wert eingegeben wurde.

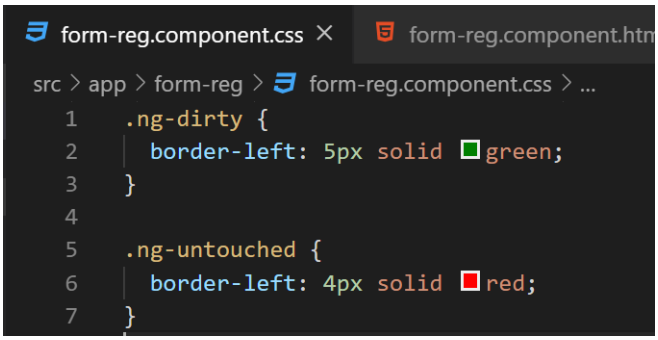
- Oder soll das Feld grün unterlegt sein, wenn der Nutzer eine Eingabe vorgenommen hat.

4.1.)Man kann eigene Klassen in den „CSS“-Code schreiben.

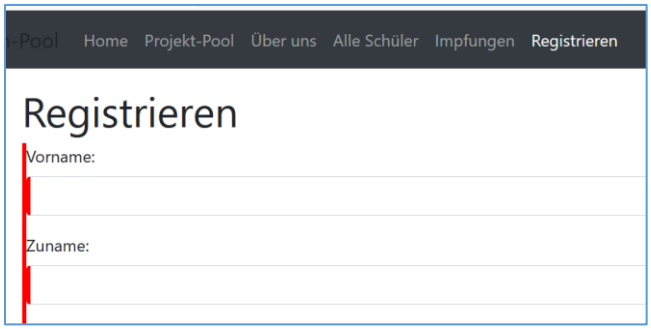
Gib in die „form-reg.component.css“ folgenden Code ein:

```
 .ng-dirty {
  border-left: 4px solid green;
}

.ng-untouched {
  border-left: 4px solid red;
}
```



Das Ergebnis ist aber nicht ganz zufriedenstellend: es ist schon bevor ein User was eingibt das „abschreckende“ Rot vorhanden. Auch der Rote Strich ganz außen ist nicht gewünscht.



Lösche den CSS-Code wieder.

4.2.)CSS Klassen von Angular nutzen (ohne eigenen CSS-Code)

Beispiel: Inputfeld - Vorname

Sinn: der Vorname muss eingegeben werden und soll bei Nichteingabe ein sichtbares Zeichen bekommen, dass hier das nicht ordnungsgemäß gemacht wurde (es ist somit „invalid“).

Man muss das Feld eingeben:

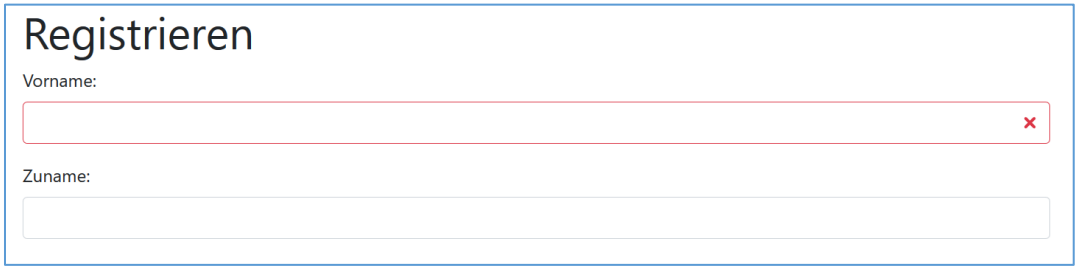
- required

Schreibe in die Klasse folgenden Code

- is-invalid

```
5    <div class="form-group">
6      <label>Vorname:</label>
7      <input type="text" name="vorname" required  class="form-control is-invalid" ngMo
8    </div>
```

Ergebnis:



Ein roter Rahmen (der ist automatisch durch Angular vorhanden) ist bereits von Beginn an vorhanden. Das bedeutet, dass ein User sofort „abgeschreckt“ ist, das soll eigentlich vermieden werden. Sinnvoll wäre es, nicht gleich beim Laden des Formulars den Rahmen anzuzeigen, sondern erst nachdem das Feld berührt wurde und es leer gelassen wurde.

Dafür kann man für ein Feld mehrere Validierungen kombinieren:

- required
- minlength=“6“
- maxlength=“20“
- pattern=“[a-z]“ …hier werden nur Kleinbuchstaben akzeptiert
- pattern=“\d{10}“ …es müssen genau 10 Zeichen sein
- pattern=“.*\d.*“ …es muss mindestens eine Zahl beinhalten
- email …das Feld muss eine E-Mail Adresse beinhalten

Folgender Code:

- zuerst mit Raute einen beliebigen, passenden Namen geben und ngModel dazu

```
6      <label>Vorname:</label>
7      <input type="text" name="vorname" required #vorname="ngModel"  class="form-cont
8    </div>
```

- man verwendet „class-binding“
  in eckigen Klammern „class“ mit Punkt danach und die gewünschte CSS-Klasse, die man gleich von vorhin ausschneiden kann und hier einfügen kann.

```
vorname" required  class="form-control is-invalid" ngM
```

```
6      <label>Vorname:</label>
7      <input type="text" name="vorname" required #vorname="ngModel"  [class.is-invalid] cl
8    </div>
```

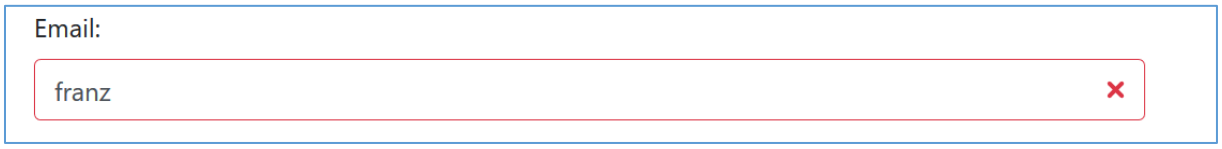
  Nun nutzt man die Referenz zu #vorname=“ngModel“ und fügt das nach den Istgleichzeichen an und checked dann die „invalid-Property“:
  Aber der Rahmen ist noch vorhanden, wenn man die Seite lädt:
  ZIEL: Der roten Rahmen soll NICHT angezeigt werden, wenn die Seite geladen wird, erst dann, wenn eine Eingabe nicht den Kriterien entspricht.
  Daher erweitere das Klassen-Binding um ein weiteres Kriterium, nämlich ob das Feld berührt wurde.
  Somit ist dann das Feld „invalid“, wenn es invalid und auch berührt wurde.

```
6
7   odel"  [class.is-invalid]="vorname.invalid && vorname.touched" class="form-control" ngM
8
```

Es muss also angeklickt aber dann leer gelassen worden sein, damit die Fehlermeldung erscheint. Sie erscheint somit nicht mehr beim Laden der Seite.

Übung: Validierung des E-Mail Feldes

Ziel: Fehlermeldung bei „invalid“-ausgefülltem E-Mailfeld:



Lösung:

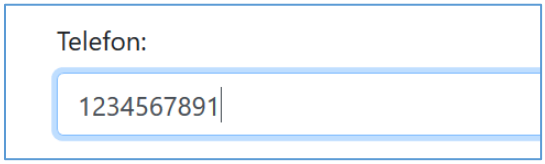
```
15    <div class="form-group">
16      <label>Email:</label>
17      <input type="email" name="email" email #email="ngModel"
18      [class.is-invalid]="email.invalid && email.touched" class="form-control" ngModel>
19    </div>
20
```

Übung: Telefonnummer muss genau 10 Zeichen haben = pattern="^\d{10}$"

```
21    <div class="form-group">
22      <label>Telefon:</label>
23      <input type="tel" name="telefon" class="form-control" ngModel
24      #telefon="ngModel" pattern="^\d{10}$"
25      [class.is-invalid]="telefon.invalid && telefon.touched">
26    </div>
```

Code:
#telefon="ngModel" pattern="^\d{10}$" [class.is-invalid]="telefon.invalid && telefon.touched">

Ergebnis:



Info: youtube, ab ca. 5:00 Minuten: https://www.youtube.com/watch?v=kFHkztHm8xA&list=PLC3y8-rFHvwhwL-XH04cHOpJnkgRKykFi&index=8

5)Anzeigen von Fehlermeldungen

Für einen Benutzer ist es sinnvoll zu erfahren, warum ein Feld „invalid“ ist und einen roten Rahmen erhalten hat.

Dafür soll eine passende Meldung, in kleinerer Schrift (small) unter dem Feld angezeigt werden, und am besten auch in der gleichen roten Farbe wie der Rahmen.

Eine einfache gute Lösung bietet Bootstrap 4. Dabei verwendet man die Klasse „d-none“, die sozusagen für das „Verstecken von Elementen“ (hiding elements) zuständig ist. Das „d-none“ steht hier für „display-none“.

https://getbootstrap.com/docs/4.3/utilities/display/#hiding-elements

5.1)Beispiel: Vorname, der „required“ ist.

Ziel: die Meldung unterhalb des Feldes soll versteckt sein, solange das Feld „valid“ und „unberührt“ ist.

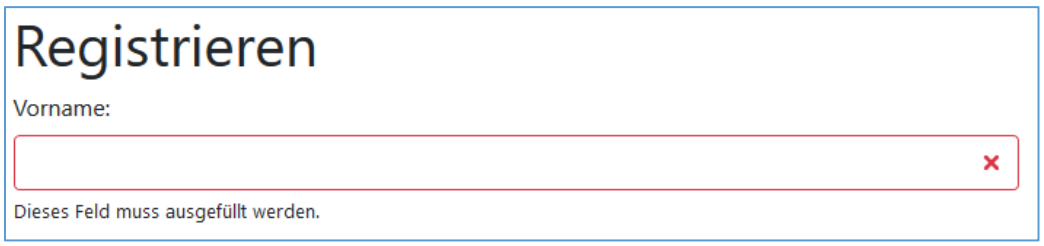
Dazu verwendet man hier class-binding mit eckigen Klammern, darin die Klasse „class.d-none“, gefolgt von zwei Konditionen mit einer ODER Verbindung.

Code:

<small [class.d-
none]="vorname.valid || vorname.untouched">Dieses Feld muss ausgefüllt werden. </small>

```
5     <div class="form-group">
6       <label>Vorname:</label>
7       <input type="text" name="vorname" required #vorname="ngModel"
8       [class.is-invalid]="vorname.invalid && vorname.touched"
9       class="form-control" ngModel>
10      <small [class.d-none]="vorname.valid || vorname.untouched">Dieses Feld muss
11        ausgefüllt werden. </small>
12    </div>
13
```

Ergebnis:

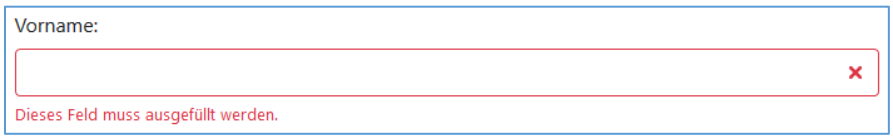


Die Farbe auf rot stellen, damit die Meldung wie eine Fehlermeldung aussieht.

Dazu verwende die einfache Bootstrap 4 Klasse: class="text-danger"

```
<small class="text-danger" [class.d-none]="vorname.valid || vorname.untouched">
  Dieses Feld muss ausgefüllt werden. </small>
```

Ergebnis:



5.2)Beispiel Telefonnummer muss 10 Zeichen haben

Starte mit der Meldung in <small>:

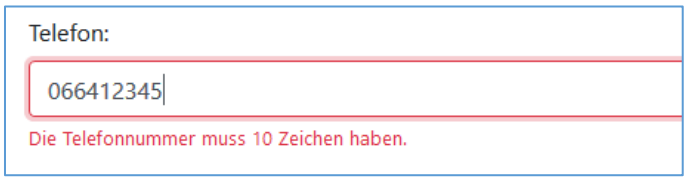
```
25      <div class="form-group">
26        <label>Telefon:</label>
27        <input type="tel" name="telefon" class="form-control" ngModel
28        #telefon="ngModel" pattern="^\d{10}$"
29        [class.is-invalid]="telefon.invalid && telefon.touched">
30        <small> Die Telefonnummer muss 10 Zeichen haben.</small>
31      </div>
```

Dann der gleiche Code wie oben beim Beispiel:

```
<small class="text-danger" [class.d-none]="telefon.valid || telefon.untouched">
  Die Telefonnummer muss 10 Zeichen haben.</small>
```

Code:
<small class="text-danger" [class.d-none]="telefon.valid || telefon.untouched">

Ergebnis:



Info:
https://www.youtube.com/watch?v=U8FfRRLFmSY&list=PLC3y8-rFHvwhwL-XH04cHOpJnkgRKykFi&index=9

6)Vor dem Absenden prüfen, ob die Form insgesamt passt

https://www.youtube.com/watch?v=kiR3J3MoWV4&list=PLC3y8-rFHvwhwL-XH04cHOpJnkgRKykFi&index=11

Ziel: der Submit-Button soll erst auslösbar sein, wenn das Formular korrekt ausgefüllt ist.

Mit anderen Worten: er ist solange „disabled“ solange das Formular „invalid“ ist.

Daher muss dies in den Submit-Button gebracht werden:

Vorher:

```
64
65        <button class="btn btn-primary" type="submit">Registrieren</button>
66    </form>
```

Wird zu:

```
65      <button [disabled]="regForm.form.invalid" class="btn btn-primary"
66      type="submit">Registrieren</button>
67  </form>
```

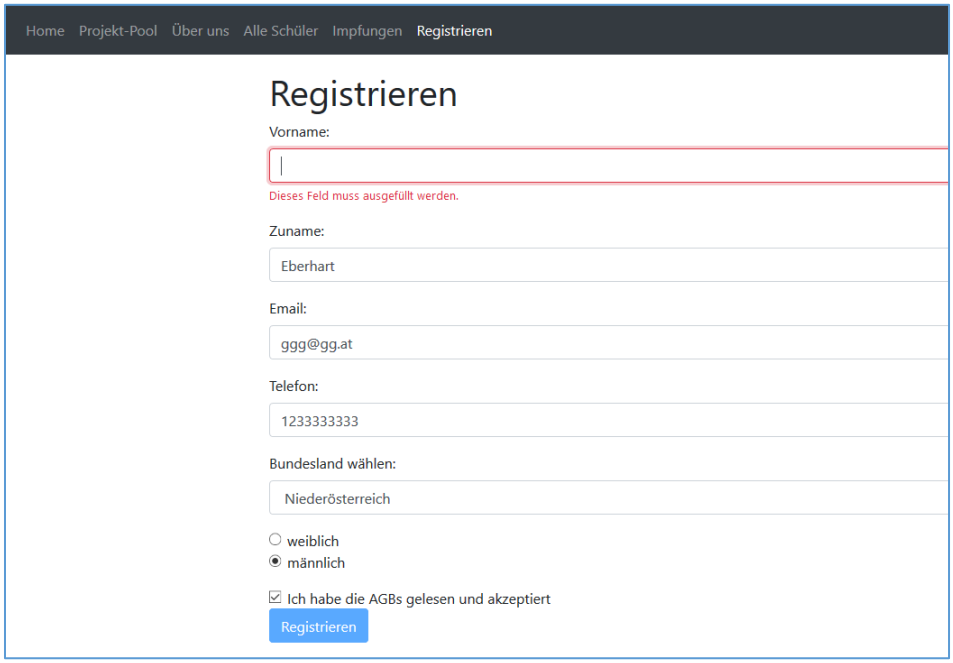
Man spricht von „disabled property binding“.

Der Name „regForm“ kommt aus der Zeile 3 ganz oben:

form-reg.component.html

```
src > app > form-reg > form-reg.componer
1   <br><br><br>
2     <h1>Registrieren</h1>
3       <form #regForm="ngForm">
```

Ergebnis: der Button kann nicht geklickt werden, da der Vorname fehlt

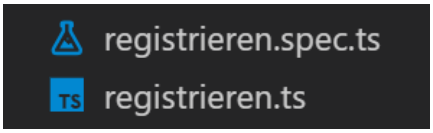


7)Daten an ein Model binden, um es an einen Server senden zu können

7.1)Man muss zuerst ein Model in einer Klasse erstellen

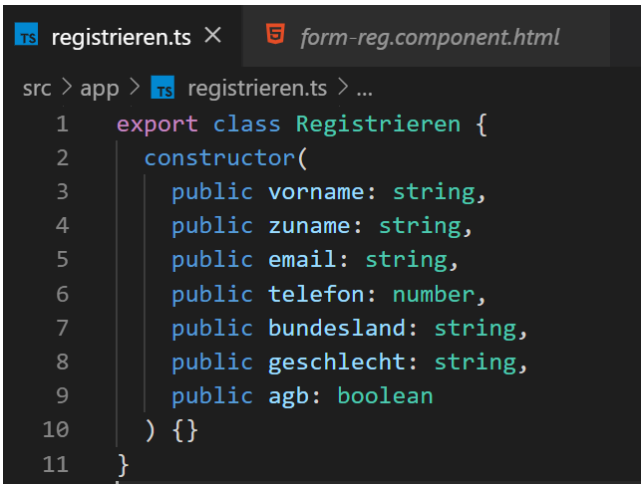
ng g class registrieren

Ergebnis:



Öffne „registrieren.ts“.

Lege folgenden Konstruktor an:



Code:

```
constructor(
   public vorname: string,
   public zuname: string,
   public email: string,
   public telefon: number,
   public bundesland: string,
   public geschlecht: string,
   public agb: boolean
 ) {}
```

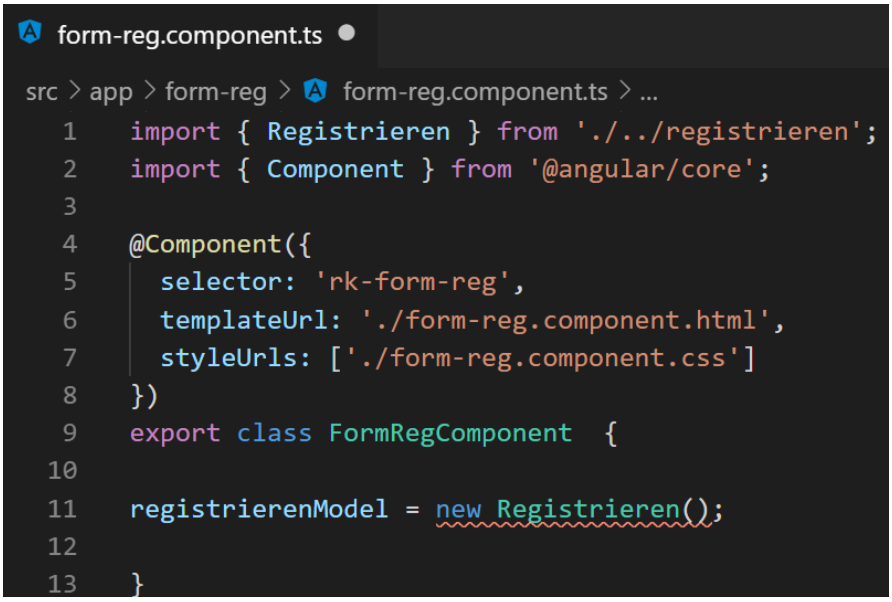
7.2)Eine Instanz in der zugehörigen ts-Datei anlegen

Öffne die „form-reg.component.ts“

Vorhanden ist folgendes, wobei man den „constructor“ du „ngOnInit“ löschen kann, ebenfalls das „OnInit“ in Zeile 1 und in Zeile 8:

Danach:

Hier folgt nun nach der Zeile 8 im „export“:



Der „import“ in Zeile 1 erfolgt automatisch bzw. händisch.

7.3)Data binding im Formular (Html)

Dafür wird nun jedes Formularfeld ein Two-Way-Binding erhalten.

Das „ngModul“ erhält eckige und runde Klammern für das „Two-Way-Data-Binding“. Damit werden neue Werte SOFORT in das „schuelerklasse“-Objekt übertragen.

Verwende das bereits vorhandene „ngModel“ und umschließe es mit den beiden Klammern vorne und hinten (sogenannte Banana in the Box). Dann das eben in der „ts-Datei“ angelegte Model und den Namen des Feldes, indem man dieses Binding durchführt:

[(ngModel)]="registrierenModel.vorname"

```
<div class="form-group">
  <label>Vorname:</label>
  <input type="text" name="vorname" required #vorname="ngModel"
  [class.is-invalid]="vorname.invalid && vorname.touched"
  class="form-control" [(ngModel)]="registrierenModel.vorname">
  <small class="text-danger" [class.d-none]="vorname.valid || vorname.untouched">
    Dieses Feld muss ausgefüllt werden. </small>
</div>
```

Das für jedes einzelne Feld:

```
<div class="form-group">
  <label>Zuname:</label>
  <input type="text" name="zuname" class="form-control"
   [(ngModel)]="registrierenModel.zuname">
</div>
```

```
<div class="mb-3">
  <div class="form-check">
    <input class="form-check-input" [(ngModel)]="registrierenModel.geschlecht"
    type="radio" name="geschlecht" value="weiblich">
    <label class="form-check-label">weiblich</label>
  </div>
  <div class="form-check">
    <input class="form-check-input" [(ngModel)]="registrierenModel.geschlecht"
    type="radio" name="geschlecht" value="männlich">
    <label class="form-check-label">männlich</label>
  </div>
</div>
```

Ergebnis: das Formular ist nun vorausgefüllt und funktioniert immer noch