**ANGULAR FORMS**

**First steps:**

|  |
| --- |
| **Use for all *template driven* exercises: /forms-template-driven, Component0**  **Use ‘npm install’, to install the dependencies**  **Use ‘ng serve’, to start this application at** [**http://localhost:4200**](http://localhost:4200) |

**0) Make Component0 active as a route:**

* **Add this Component0 to the router (app.routes.ts)**
* **Import and define the Component0 in the app Module (app.module.ts).**
* **And add a routerLink in the MainComponent for this Component0.**

**1) Goal: build a template driven form**

o Create a simple HTML5 form in Component0.

* Import FormsModule into your app.module.ts

o Add the local template variable #myForm="ngForm" to the <form> tag.  
o Add the directives ngModel to the separate form fields. You don't need two‐way databinding with [()].  
o Write for example myForm.value to the user interface or show the contents of the form in an alert (or in the console) when a button is clicked.  
o Demo code available at **/forms-template-driven**, Component 1.

**2) Goal: address individual controls inside the form and add HTML5 validators**

o Assign a **local template variable** to the form fields.  
o Bind ngModel to the **local template variable**. The code can look like: #email="ngModel"  
o Retrieve the values from the **local template variable** and show them in the user interface, for example its value and its validity.  
o Add the HTML5 attribute required to the form fields and see how this affects the state of the form field. Write its validity to the user interface.  
o Demo code available at **/forms-template-driven**, Component 2.

**3) Goal: combining individual form fields to an ngModelGroup**

o Add some field to the form (for example some extra text fields, selectbox or checkboxes ).  
o Group them inside a <div>, assign the <div> the directive ngModelGroup. The code can look like:

<div ngModelGroup="customer" #customer="ngModelGroup">  
o Run the code and identify the model group in the returned form value object.  
o Optional: set the value of a form field from inside your class, by using the local template variable and bind to [ngModel].  
o Demo code available at **/forms-template-driven**, Component 3.

**4) Goal: submitting template driven forms**

o Add a submit button to the form.  
o Make sure the submit button is only active when the form as a whole is valid. Your code can look like:

<button type="submit"

(click)="onSubmit(myForm)"

[disabled]="!myForm.valid">

...

</button>

o Demo code available at **/forms-template-driven**, Component 4.

**5) Goal: working with model driven forms**

o Start with a simple form, for example build a form with the following elements:

* Username
* Surname
* *Telephone* number
* BirthDate
* Email
* Gender (m.b.v. Radio-button)
* DeveloperSkillLevel (dropdown)

o Import ReactiveFormsModule into your app.module.ts.

and Import FormGroup, FormBuilder and AbstractControl into your app.component.ts.

import {FormGroup, FormBuilder, Validators, AbstractControl} from '@angular/forms';

o Add the [formGroup]="..." directive to the <form> tag, add formControlName="..." to the individual controls.  
o Import FormGroup and FormBuilder into your class and build the form, based on the layout of your HTML.  
o Submit the form and write the value to an alert box or to the console.  
o Demo code available at **/forms-model-driven, Component 1**

**6) Goal: add validation to model driven forms**

o Import Validators into your app.component.ts.

import {FormGroup, FormBuilder, Validators, AbstractControl} from '@angular/forms';

o Add ‘default’ validations to the form fields.

o Add ‘custom’ validations to the form fields.

* Validate your *email pattern, unique username (names from an array), telephone number (min 10, max 15).*

Demo code available at **/forms-model-driven, Component 2**

**7) unittest reactive forms**

See: <https://codecraft.tv/courses/angular/unit-testing/model-driven-forms/>

**8) FormArray**

Add an address list to the existing form. Each address has:

* Street
* House number
* Postal Code

Also ensure that the user can add **new addresses** and remove existing addresses.

Write an AddressValidator. Requirements: minimal 2 addresses and a unique Postal Code.

And validate the *postal Code (regex: /^[1-9][0-9]{3} ?(?!sa|sd|ss)[a-z]{2}$/i )*

**9) Optional:**

Add/change **rxjs/operators** to the Demo app: /forms-typeahead, Component 1

Use alternative Live Api’s. See file: JavaScript APIs.txt