# Angular Best Friends

## Module 7 Exercise 1 – Reactive form

## Goal

In this exercise we’ll want to better understand the difference between template driven forms and reactive forms by converting the previously demoed form into a reactive form.

## Steps

1. The initial status of the exercise is placed in the **Module7Exercise1 > initial** folder in the **Angular friends** repository.
2. Open it in VS Code. From the terminal run **npm install** to install the needed node modules.
3. We’ll then start from the component class in **customer.component.ts.**
4. In the save method we won’t need any incoming parameters. So delete it. It should look like this afterwards:

save() {

console.log(this.customerForm);

console.log('Saved: ' + JSON.stringify(this.customerForm.value));

}

1. At the top of the file add these imports:

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

1. Before or after the “customer” property add another one of type FormGroup. At the end it should look like this:

customerForm: FormGroup;

customer = new Customer();

1. Import the FormBuilder service into the constructor. It should look like this:

constructor(private fb: FormBuilder) { }

1. In ngOnInit we’ll create a new instance of FormGroup and assign it to the property we created at step 6. The best way to proceed is to delete the entire ngOnInit function and replace it with this one:

ngOnInit() {

this.customerForm = this.fb.group({

firstName: '',

lastName: '',

email: '',

sendCatalog: true

});

}

So we use the FormBuilder class to create a new form group and assign it to our property. On the form builder we use the group() function to initialize the needed form controls. So firstName, lastName, email and sendCatalog are all form controls created by the form builder. The sendCatalog form control has a default value set to true.

1. Under ngOnIniti add a new function called populateData(). This function will be wired to a button on the UI that we’ll be able to use to populate the form with data. From Angular perspective this is useful for us since we’ll learn how to update form control values. Here’s the function that you need to add:

populateTestData(): void {

this.customerForm.patchValue({

firstName: 'Jack',

lastName: 'Harkness',

sendCatalog: false

});

}

1. Now go to the customer.component.html file. Delete all the markup and paste the following markup instead:

<div class="card">

<div class="card-header">

Sign Up!

</div>

<div class="card-body">

<form novalidate

(ngSubmit)="save()"

[formGroup]="customerForm">

<div class="form-group row mb-2">

<label class="col-md-2 col-form-label"

for="firstNameId">First Name</label>

<div class="col-md-8">

<input class="form-control"

id="firstNameId"

type="text"

placeholder="First Name (required)"

required

minlength="3"

formControlName="firstName"

[ngClass]="{'is-invalid': (customerForm.get('firstName').touched || customerForm.get('firstName').dirty) && !customerForm.get('firstName').valid }" />

<span class="invalid-feedback">

<span \*ngIf="customerForm.get('firstName').errors?.required">

Please enter your first name.

</span>

<span \*ngIf="customerForm.get('firstName').errors?.minlength">

The first name must be longer than 3 characters.

</span>

</span>

</div>

</div>

<div class="form-group row mb-2">

<label class="col-md-2 col-form-label"

for="lastNameId">Last Name</label>

<div class="col-md-8">

<input class="form-control"

id="lastNameId"

type="text"

placeholder="Last Name (required)"

required

maxlength="50"

formControlName="lastName"

[ngClass]="{'is-invalid': (customerForm.get('lastName').touched || customerForm.get('lastName').dirty) && !customerForm.get('lastName').valid }" />

<span class="invalid-feedback">

<span \*ngIf="customerForm.get('lastName').errors?.required">

Please enter your last name.

</span>

<span \*ngIf="customerForm.get('lastName').errors?.maxlength">

The last name must be less than 50 characters.

</span>

</span>

</div>

</div>

<div class="form-group row mb-2">

<label class="col-md-2 col-form-label"

for="emailId">Email</label>

<div class="col-md-8">

<input class="form-control"

id="emailId"

type="email"

placeholder="Email (required)"

required

email

formControlName="email"

[ngClass]="{'is-invalid': (customerForm.get('email').touched || customerForm.get('email').dirty) && !customerForm.get('email').valid }" />

<span class="invalid-feedback">

<span \*ngIf="customerForm.get('email').errors?.required">

Please enter your email address.

</span>

<span \*ngIf="customerForm.get('email').errors?.email">

Please enter a valid email address.

</span>

</span>

</div>

</div>

<div class="form-group row mb-2">

<div class="col-md-8">

<div class="form-check">

<label class="form-check-label">

<input class="form-check-input"

id="sendCatalogId"

type="checkbox"

formControlName="sendCatalog"> Send me your catalog

</label>

</div>

</div>

</div>

<div class="form-group row mb-2">

<div class="offset-md-2 col-md-4">

<button class="btn btn-primary mr-3"

type="submit"

style="width:80px"

[title]="customerForm.valid ? 'Save your entered data' : 'Disabled until the form data is valid'"

[disabled]="!customerForm.valid">

Save

</button>

<button class="btn btn-outline-secondary"

type="button"

(click)="populateTestData()">

Test Data

</button>

</div>

</div>

</form>

</div>

</div>

<br>Dirty: {{ customerForm.dirty }}

<br>Touched: {{ customerForm.touched }}

<br>Valid: {{ customerForm.valid }}

<br>Value: {{ customerForm.value | json }}

1. What differences do you notice when comparing to the template driven form?
2. Now you can run the app an test the newly added functionality!