

Angular

Lab 4

Reactive Forms

1	LAB SETUP	1
2	GENERATING A NEW ANGULAR PROJECT	2
3	ADDING A BASIC FORM CONTROL USING FORMCONTROL	3
4	USING FORMGROUP TO GROUP RELATED FORM CONTROLS	4
5	SUBMITTING THE FORM AND ACCESSING FORM AND FORM CONTROL VALUES	4
6	UPDATING ALL OR A PORTION OF A FORMGROUP	5
7	CONTROL STATUS FOR A FORMCONTROL AND FORMGROUP	6
8	VALIDATING FORM CONTROLS	7
9	DISPLAY VALIDATION ERROR MESSAGES	8
10	USING FORMBUILDER AS A SHORTCUT FOR CREATING FORMGROUP	9
11	USING FORMARRAY TO BUILD DYNAMIC FORMS	9
12	OTHER FORM CONTROL ELEMENTS (RADIO BUTTONS, CHECKBOXES, ETC)	10

1 Lab setup

Make sure you have the following items installed

- Latest version of Node and NPM (note: labs are tested with Node v16.13 and NPM v8.1 but should work with later versions with no or minimal changes)
- Latest version of TypeScript (note: labs are tested with TypeScript v4.5.4 but should work with later versions with no or minimal changes)
- Latest version of Angular (note: labs are tested with Angular v13.1 but should work with later versions with no or minimal changes)
- Visual Studio Code (or a suitable alternative IDE for Angular/TypeScript)
- A suitable text editor (Notepad ++)
- A utility to extract zip files (7-zip)

In each of the main lab folders, there are two subfolders: `changes` and `final`.

- The `changes` subfolder holds the source code and other related files for the lab. The creation of the Angular app will proceed in a step-wise fashion in order to clearly demonstrate the various features of the framework. The various project source files in this folder are numbered in a way to reflect this gradual construction. For e.g. `xxx.v1.html`, `xxx.v2.html`, etc represents successive changes that are to be made to the `xxxx.html`

- The `final` subfolder holds the complete Angular project. If there was some issue in completing the step-wise construction of the app using the files from the `changes` subfolder, you can use this to run the final working version.

The project folder in `final` is however missing the crucial `node_modules` subfolder which contains all the dependencies (JavaScript libraries) that your Angular app needs in order to be built properly. These dependencies are specified in the `package.json` file in the root project folder. They are omitted in the project commit to GitHub because the number and size of the files are prohibitively large.

In order to run the app via the Angular development server (`ng serve`), you will need to create this subfolder containing the required dependencies. There are two ways to accomplish this:

- a) Copy the project root folder to a suitable location on your local drive and run `npm install` in a shell prompt in the root project folder. NPM will then create the `node_modules` subfolder and download the required dependencies as specified in `package.json` to populate it. This process can take quite a while to complete (depending on your broadband connection speed)
- b) Alternatively, you can reuse the dependencies of an existing complete project as most of the projects for our workshop use the same dependencies. In this case, copy the `node_modules` subfolder from any existing project, and you should be able to immediately start the app. If you encounter any errors, this probably means that there are some missing dependencies or Angular version mismatch, in which case you will then have to resort to `npm install`.

The lab instructions here assume you are using the Chrome browser. If you are using a different browser (Edge, Firefox), do a Google search to find the equivalent functionality.

2 Generating a new Angular project

The source code for this lab can be found in the `changes` subfolder of `Directives-Demo`

In either a separate command prompt (or shell terminal) or in an embedded terminal in Visual Studio Code, create a new Angular project with:

```
ng new reactiveformsdemo
```

Press enter to accept the default values for all the question prompts that follow regarding routing and stylesheet.

Navigate into the root project folder from the terminal with:

```
cd reactiveformsdemo
```

Build the app and serve it via Angular's live development server by typing:

```
ng serve
```

The final output line from this should indicate that the compilation process was successful and identify the port that the live development server is currently serving up the Angular app dynamically from (by default this will be port 4200)

Open a browser tab at:

<http://localhost:4200/>

You should be able to see the default landing page for all new autogenerated Angular projects.

To stop the development server, type Ctrl+C in the terminal window that it is running in. You can restart again anytime by typing `ng serve` in the project root folder.

3 Adding a basic form control using FormControl

The source code for this lab can be found in the `changes` subfolder of `ReactiveForms-Demo`

Modify the following files in `src/app` with the latest update from `changes`:

`app.component-v1.ts`

`app.component-v1.html`

`app.module-v1.ts`

There are three steps to using form controls.

1. Register the reactive forms module in your application. This module declares the reactive-form directives that you need to use reactive forms.
2. Generate a new FormControl instance and save it in the component.
3. Register the FormControl in the template.

You can display the value in the bound form control in two ways:

1. Using the `valueChanges` observable where you can listen for changes in the form's value in the template using `AsyncPipe` or in the component class using the `subscribe()` method.
2. The `value` property gives you a snapshot of the current value

References:

<https://www.tektutorialshub.com/angular/formcontrol-in-angular/#what-is-formcontrol>

<https://www.tektutorialshub.com/angular/formcontrol-in-angular/#using-formcontrol>

<https://angular.io/guide/reactive-forms#adding-a-basic-form-control>

<https://angular.io/guide/reactive-forms#displaying-a-form-control-value>

4 Using FormGroup to group related form controls

Modify the following files in `src/app` with the latest update from changes:

`app.component-v2.ts`

`app.component-v2.html`

Forms typically contain several related controls. Reactive forms provide two ways of grouping multiple related controls into a single input form: a `FormGroup` and `FormArray`.

A `FormGroup` provides a wrapper around a collection of related form elements and the `FormControls` that they are bound to. It encapsulates all the information related to this group of form elements. It tracks the value and validation status of each of these controls, and also the state of the group as a whole.

To add a `FormGroup` to a component, do the following:

1. Create a `FormGroup` instance
2. Create multiple `FormControl` instances within the main `FormGroup` instance (this can also include other nested `FormGroup` instance as well if necessary)
3. Bind the top-level `<form>` element in the template to the `FormGroup` instance
4. Bind the individual form controls in the `<form>` element to the individual `FormControl` instances using the `formControlName` directive

References:

<https://angular.io/guide/reactive-forms#grouping-form-controls>

<https://www.tektutorialshub.com/angular/formgroup-in-angular/#what-is-formgroup>

<https://www.tektutorialshub.com/angular/formgroup-in-angular/#using-formgroup>

<https://www.tektutorialshub.com/angular/formgroup-in-angular/#reactive-forms>

<https://www.tektutorialshub.com/angular/formcontrol-in-angular/#reactive-forms>

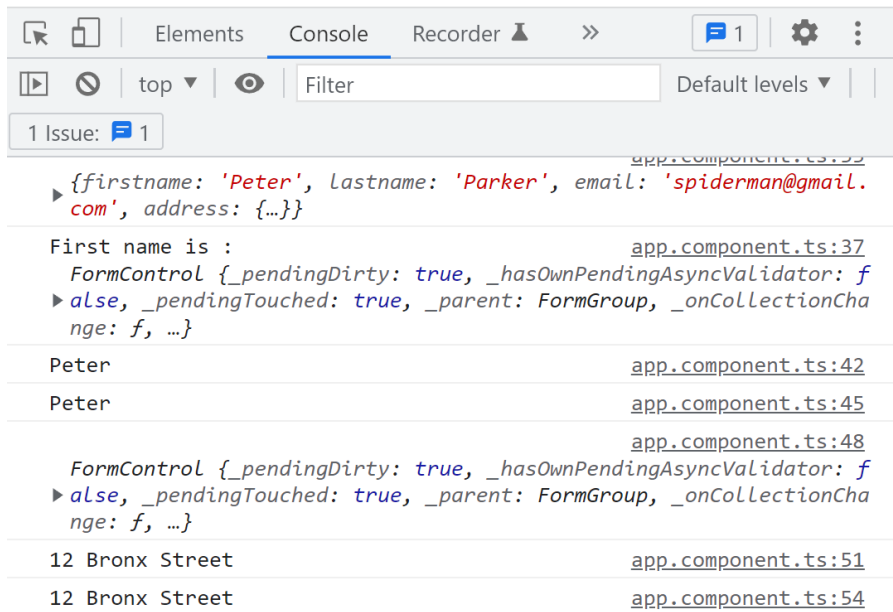
5 Submitting the form and accessing form and form control values

Modify the following files in `src/app` with the latest update from changes:

`app.component-v3.ts`

`app.component-v3.html`

Complete the form with appropriate values, click the Submit button and check for output in the DevTools Console panel.



The FormGroup directive listens for the submit event emitted by the form element when a submit type button within the form is clicked, and emits a ngSubmit event that can be bound to a component method.

We can access the value of the FormGroup as a whole which is an object whose keys are the form control names and values are the contents of the text fields. With nested form groups present, the keys are the form group names and value is an object whose keys / values are the form control names and contents of the nested elements

Notice that the FormControl object contains many additional elements which are used to reflect the status of the control: such as valid, pending, pristine, dirty, touched, etc.

References:

<https://www.tektutorialshub.com/angular/formcontrol-in-angular/#finding-the-value>

6 Updating all or a portion of a FormGroup

Modify the following files in `src/app` with the latest update from changes:

`app.component-v4.ts`

`app.component-v4.html`

We use `setValue` or `patchValue` method of the FormControl to set a new value for the form control. There is no difference between `setValue` and `patchValue` at the FormControl level.

For a FormGroup, the `setValue` method call on it requires that all the properties of the object passed to it matches all the FormControl instances within that FormGroup, otherwise an error occurs.

For a FormGroup, the patchValue method call is used to update only a subset of the FormControl instances within it, while ignoring the rest.

We can reset or clear the contents of all FormControl instances within the FormGroup with the reset method call.

References:

<https://www.tektutorialshub.com/angular/formcontrol-in-angular/#setting-the-value>

<https://angular.io/guide/reactive-forms#updating-parts-of-the-data-model>

<https://www.tektutorialshub.com/angular/formgroup-in-angular/#setting-value>

<https://www.tektutorialshub.com/angular/setvalue-patchvalue-in-angular/>

7 Control status for a FormControl and FormGroup

Modify the following files in `src/app` with the latest update from changes:

`app.component-v5.ts`

`app.component-v5.html`

Try typing into the First Name field, and clicking in and out of the field without changing anything to see the effect on the various properties of the firstname control. Repeat this for any other field (other than the First Name field) to see the effect on the various properties of the entire FormGroup. You will need to refresh the app to restore the properties to their original values when experimenting

The FormControl object holds properties related to the status of the element that it is bound to. Some of this are related to validation of the FormControl, which we will examine in the next section. Besides validation, there are 2 other important properties: pristine/dirty and touched/untouched.

- Pristine - This is true if the user has not yet changed the content of the element.
- Dirty - This is true if the user has changed the content of the element.
- Touched - This is true if the user has clicked at least once inside the element and then clicked away, even if nothing is changed (or triggered a blur event)
- Untouched - This is true if the user has not interacted with the element in any way

The FormGroup object also has the same properties, and these properties will have the same values as explained above as long as any one of the FormControl objects in that FormGroup object has that value as well. For e.g. a FormGroup is considered to be dirty as long as any one of the FormControl objects within it is considered dirty.

References:

<https://www.tektutorialshub.com/angular/formcontrol-in-angular/#control-status>

<https://www.tektutorialshub.com/angular/formgroup-in-angular/#control-status>

8 Validating form controls

Validating the value of form controls is a very important aspect of web application development in order to ensure that invalid data is not accidentally accepted and stored / processed in the system as this can cause problems for the core business logic of the application. Form validation can be performed client-side by Angular (or any other JavaScript program / framework), and also server side: and it is typically recommended to perform form validation on both sides to avoid the possibility of hackers circumventing the client-side validation functionality

Modify the following files in `src/app` with the latest update from changes:

`app.component-v6.ts`

`app.component-v6.html`

Reactive forms use validator functions to perform their validation functionality. These are added directly to the FormControl objects in the FormGroup. You can use any of the standard built-in validators that come out of the box from Angular ([Angular - Validators](#)) or create your own custom validator. Validator functions can either be synchronous (the most common approach) or asynchronous.

The main property for a FormControl object related to validation is status. Based on the result of the validation checks by Angular, the property can have four possible values.

- VALID: The FormControl has passed all validation checks.
- INVALID: The FormControl has failed at least one validation check.
- PENDING: The FormControl is in the midst of conducting a validation check.
- DISABLED: The FormControl is exempt from validation checks

In addition, there are related properties such as

- valid - true when the value of the FormControl status is VALID (there is also an invalid property with the opposite meaning)
- pending - true when the value of the FormControl status is PENDING
- disabled - true when the value of the FormControl status is DISABLED (there is also an enabled property with the opposite meaning)

The FormGroup object also has the same properties. Its status property will be VALID if every single FormControl within it is valid, and will be INVALID otherwise. Same comments apply for its valid property.

HTML also provides a variety of [attributes for input elements that are related to validation](#), so you can use this as well. Examples include required, size, maxlength, min, max, pattern.

If you use HTML-related validation attributes, then the validation functionality is enforced by the browser, whereas validation functionality provided via validator functions are enforced by Angular. Mixing both approaches in a single form can result in inconsistency on how invalid data is handled, so sometimes we may wish to disable the HTML validation functionality using the `novalidate` attribute added to the main overall `<form>` element and allow Angular to handle the validation process completely.

References:

<https://angular.io/guide/form-validation#validating-input-in-reactive-forms>

<https://www.tektutorialshub.com/angular/angular-reactive-forms-validation/#validators-in-reactive-forms>

For full list of Validators:

<https://angular.io/api/forms/Validators>

9 Display validation error messages

Modify the following files in `src/app` with the latest update from `changes`:

`app.component-v7.ts`

`app.component-v7.html`

We can add in suitable validation error messages to inform the user using the `NgIf` directive. The conditional logic for the directive will therefore incorporate the values of the `valid` / `invalid` properties

`app.component-v7-2.ts`

`app.component-v7-2.html`

Typically we will add in getter methods in the component class to make the syntax for accessing the required `FormControl` instances in the template easier to write. The code logic remains exactly identical in this case. We can also additionally disable the Submit button in the event that any form field is invalid.

`app.component-v7-3.html`

We can also incorporate other properties as well, such as `dirty` and `touched`, to fine tune the logic behind the display of the validation error messages. For e.g. we would typically not want to display any validation error messages when the form is loaded in an empty state for the first time. Only when the user interacts with the form field (either by specifically typing into it or clicking in it and clicking away), do we then display the error message.

`app.component-v7-4.html`

At the moment, we are displaying a generic error message of the form: `xxxx is invalid`. For `FormControls` that have 2 or more `Validator` functions associated with it, we can use the `errors` object to determine the specific validation error and display a suitable error message. We can also style the error messages accordingly at this point.

`app.component-v7-5.css`

Angular automatically mirrors many control properties onto the form control element as [CSS classes](#). We can use these classes to style form control elements according to the state of the individual FormControls.

References:

<https://angular.io/guide/form-validation#built-in-validator-functions>

<https://www.tektutorialshub.com/angular/angular-reactive-forms-validation/#displaying-the-validationerror-messages>

10 Using FormBuilder as a shortcut for creating FormGroup

Modify the following files in `src/app` with the latest update from changes:

`app.component-v8.ts`

We can use the FormBuilder service to simplify the syntax of creating a FormGroup with multiple FormControls. We need to:

- a) Import the FormBuilder class.
- b) Inject the FormBuilder service.
- c) Generate the form contents.

References:

<https://angular.io/guide/reactive-forms#using-the-formbuilder-service-to-generate-controls>

<https://www.tektutorialshub.com/angular/angular-formbuilder-in-reactive-forms/>

11 Using FormArray to build dynamic forms

FormArray is an alternative to FormGroup for building dynamic forms where the number of form controls are not known in advance, and there is flexibility required in order to dynamically insert and remove controls. Although this can be done with a FormGroup, the FormArray instance offers more flexibility because it you don't need to define a key for each control by name.

Modify the following files in `src/app` with the latest update from changes:

`app.component-v9.ts`

`app.component-v9.html`
`app.component-v9.css`

Play around with adding in new job info via the Add info for new job button and verify that the form data is correctly contained in the main FormGroup when you submit the form. Verify that you can also

remove form groups for existing jobs, and that the form data is correctly reflected when you submit the form.

We use the FormBuilder array method to create a FormArray which we include within the main FormGroup. This FormArray will hold FormGroup items, where each FormGroup item consists of individual FormControl. We then use the FormArrayName directive within the template to bind a <div> containing the FormGroup and its corresponding FormControl elements to the FormArray object in the main FormGroup.

We can then use the *ngFor directive to iterate over each FormGroup item from within the main FormArray instance.

Modify the following files in `src/app` with the latest update from changes:

`app.component-v9-2.ts`

`app.component-v9-2.html`

Play around with adding in new profiles and deleting existing profiles in the same way that you did previously for new jobs. The ability to add new jobs and removing existing ones for a given profile is still maintained. Verify again that the form data is correctly reflected when you submit the form.

Here, we have built a form with a FormArray as the top level element within the main FormGroup. Each element within this FormArray is a FormGroup, which in itself contains a nested FormArray whose elements in turn are other FormGroups.

In the template, we will need to use two *ngFor directives to iterate over the items in the top level FormArray, as well as the items in the nested FormArray of each element that is contained in the top level FormArray.

This adds to the level of complexity in both the component and its template, but provides a high level of flexibility in creating dynamic form elements.

References:

<https://angular.io/guide/reactive-forms#creating-dynamic-forms>

<https://www.tektutorialshub.com/angular/angular-formarray-example-in-reactive-forms/>

<https://www.tektutorialshub.com/angular/nested-formarray-example-add-form-fields-dynamically/>

12 Other form control elements (radio buttons, checkboxes, etc)

Modify the following files in `src/app` with the latest update from changes:

`app.component-v10.ts`

`app.component-v10.html`

We can use `ngFor` to iterate through an array content to provide the various options for a radio buttons and drop down lists. We can also bind the specific element to a `FormControl` instance in the main `FormGroup` using the `formControlName` directive in the usual manner.

For checkboxes that support multiple options, the `FormControl` that correspond to each check box will have a Boolean value (true for checked, false for otherwise). We then need to create a `FormArray` containing a collection of Boolean `FormControls`. We also need to map these collection of `FormControls` to a collection of data items representing the actual values that are selected when specific checkboxes are ticked. Then we need to include additional logic to determine from the values of the collection of Boolean `FormControls`, which values were actually selected.

References:

<https://www.netjstech.com/2020/10/radio-button-in-angular-form-example.html#reactiveRadioBtn>

<https://www.netjstech.com/2020/10/checkbox-in-angular-form-example.html#CheckboxReactive>

<https://tutorialsforangular.com/2021/01/24/using-value-vs-ngvalue-in-angular/>