Angular
Lab 4
Reactive Forms

1 Lab setup

The setup here is identical to that in Lab 1

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 stylesheet format (CSS) and enabling Server side rendering (No)

```
G:\temp\workshoplabs>ng new bindingdemo
? Which stylesheet format would you like to use? CSS             [
https://developer.mozilla.org/docs/Web/CSS                      ]
? Do you want to enable Server-Side Rendering (SSR) and Static Site
Generation (SSG/Prerendering)? (y/N)
```

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`

There are three steps to using form controls.

1. Register the ReactiveFormsModule in the component whose template the form is going to appear in (in this case, the root component AppComponent). This module declares the reactive-form directives that you need to use reactive forms.
2. Generate a new FormControl instance 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

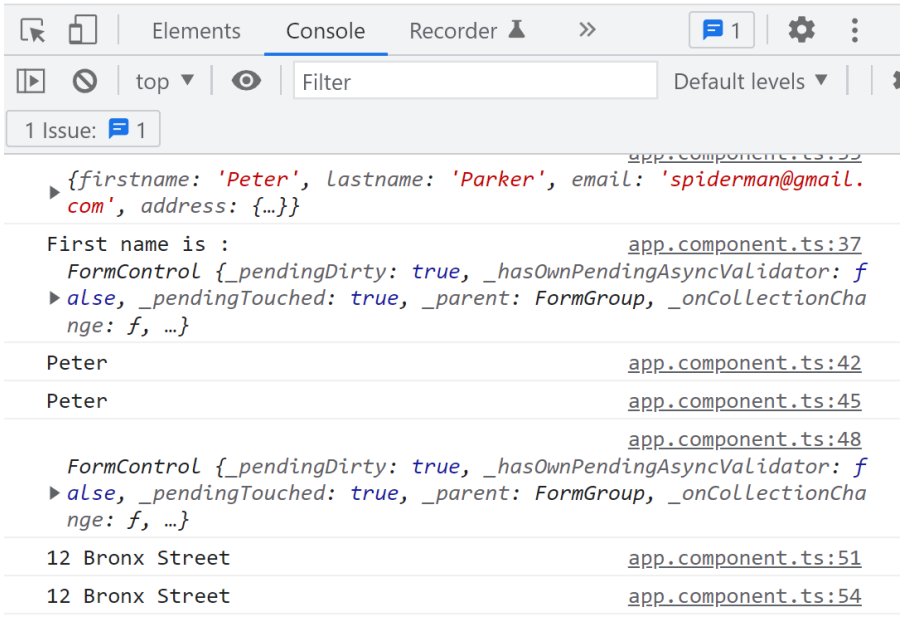
5 Submitting the form and accessing form and form control values

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

In a typical real life front end application, the component method that is bound to ngSubmit will extract the relevant content from the form controls, perform any required preprocessing, and then submit this data to the back end application. Interaction with a backend service via HTTP REST API calls will be covered in a subsequent lab. For this lab, we will just output the relevant content that we want to inspect to the Console view of the Developer Tools.

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.

You can see these properties appearing as class values for each of the individual form controls as well as the overall FormGroup in the Elements view of the Console Developer tools.

```
<h1 _ngcontent-ng-c2637375619>Reactive Forms Demo</h1>
▼<form _ngcontent-ng-c2637375619 novalidate ng-reflect-form="[object Object]"
class="ng-untouched ng-pristine ng-valid">
  ▼<p _ngcontent-ng-c2637375619>
     <label _ngcontent-ng-c2637375619 for="firstname">First Name </label>
     <input _ngcontent-ng-c2637375619 type="text" id="firstname" formcontrolname="f
     irstname" ng-reflect-name="firstname" class="ng-untouched ng-pristine ng-vali
     d">
```

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. 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. This 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 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 @if directive (applicable only for Angular 17 projects and later). The conditional logic for the directive will therefore incorporate the values of the valid / invalid properties

9.1 Simplifying access to FormControl objects in template via getter methods

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

`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.

9.2 Finetuning logic for displaying validation error messages

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

`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.

9.3 Using errors object to determine validation error

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

`app.component-v7-4.html`

`app.component-v7-4.css`

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 by applying appropriate class values to the elements containing the error messages and targeting them with appropriate style rules.

9.4 Using class values on form control elements for further styling

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

`app.component-v7-5.css`

As we saw earlier in the Elements tab of the Developer tools, the various properties such as invalid, touched, pristine, dirty etc appear as class values of the various form control elements as well as the form group as a whole.

We can then 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.

Using FormBuilder means we can drop the need to import and use FormControl and FormGroup classes.

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 FormControls. We then use the FormArrayName directive within the template to bind a containing the FormGroup and its corresponding FormControl elements to the FormArray object in the main FormGroup.

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

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 Misc 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 @for to iterate through an array content to provide the various options for radio buttons and checkboxes . 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. Then we need to implement additional logic to determine from the values of the collection of Boolean FormControls, which values of the original array content 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/