Basic Forms in Angular

Introduction

- Forms are where we get the majority of our rich data input from users
- Forms can be very complex:
  - Form inputs are menat to modify data, both on the page and the server
  - Changes often need to be reflexted elsewhere no the page
  - Users have a lot of leeway in what they enter, so you need to validate values
  - The UI needs to clearly state expectations and errors, id any
    Dependent fields can have complex logic
  - We want to be able tot test our forms, without relying on DOM selectors

Angular Form Tools

- Angular help with all of these things:
  - FormControls encapsulate the inputs in our forms and give us objects to work with them
  - Validators give us the ability to validate inputs
  - Observers let us watch our form for changes and respond accordingly

Form Controls

- A FormControl represents a single input field
- Forms Controls encapsulate the field's value, and states such as being valid, dirty (changed), ot has errors

```
let nameControl = new FormControl("Nate");
let name = nameControl.value // -> Nate

nameControl.errors // -> StringMap<string, any> of errors
nameControl.dirty // -> false
nameControl.valid // -> true
```

- We can attach tot eh DOM with an attribute, like for instance the following code:

```
<input type="text" [formControl]="name">
```

Form Groups

- Most forms have more than one field, so we need a way to manage multiple FormControls.
- Here's how you can create a FormGroup:

```
let personInfo = new FormGroup({
    firstName: new FormControl("Nate"),
    lastName: new FormControl("Murray"),
    zip: new FormControl("90210")
})
```

Form Groups (2)

- FormGroup and FormControl have a common ancestor (AbstractControl). That means we can check the status or value of personInfo just as easily as a single FormControl:

```
personInfo.value;
personInfo.errors;
personInfo.dirty;
personInfo.valid;
```

Two different kind of forms

- Reactive Forms
  - Provide direct, explicit access to the underlying form's object model. This is the reccomended type of forms in Angular.
- Template-driven Forms
  - Rely on directives in the template to create and manipulate the underlying object model. They are useful for adding a simple form to an app, such aas an email list signup form. If you have very basic form requirements, template-driven forms could be a good fit.

Key Differences for Reactive and Template-Driven

- Setup of form model:
  - reactive forms are created in component class, meanwhile template drive are created in directives
- Data model:
  - reactive forms are created by structured and immutable data, meanwhile template driven are unstructured and mutable
- Data Flow:
  - reactive forms are synchronous, template-driven forms are asynchronous
- Form validation:
  - form validation are made by functions in Angular, meanwhile in the template-driven forms, the validation is realized by directives.

An example of Reactive Forms

```
import {Component} from '@angular/core';
import { FormControl, ReactiveFormsModule } from '@angular/forms';

@Component({
    standalone: true,
    selector: 'app-reactive-favorite-color',
    template: `
        Favorite Color: <input type="text" [formControl]="favoriteColorControl">
    `,
    imports: [ReeactiveFormsModule],
})
export class FavoriteColorComponent {
    favoriteColorControl = new FormControl('');
}
```

Example of template-drive forms

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

@Component({
    standalone: true,
    selector: 'app-template-favorite-color',
    template: `
        Favorite Color: <input type="text" [(ngModel)]="favoriteColor" />
    `,
    imports: [FormsModule],
})
export class FavoriteColorComponent {
    favoriteColor = '';
}
```

Form Builder

- Form Builder is an aptly-named helper class that helps us build forms.
- Forms are made up of FormControls and FormGroups.
- FormBuilder helps us make FormControls and FormGroups.
- FormBuilder can be used for creating Reactive Forms.

Input Validations

- Our users aren't always going to enter data in exactly the right format. If someone enters data in the wrong format, we want to give them feedback and not allow the form to be submitted.
- For these reasons, we will use the input validations.

Watching for Changes

- So far we've only extracted the value from our form by calling onSubmit when the form is submitted. But often we want to watch for any value changes on a control
- Both FormGroup and FormControl have an eventemitter that we can use to observe changes.
- For doing these:
    - get access to the EventEmitter by calling control.valueChanges and then
    - add an observer using the .subscribe method

ngModel

- NgModel is a special directive: it binds a model to a form. ngModel is special in that it mimics two-way data binding.
- Two-way data binding is almost always more complicated and difficult to reason about vs. one-way data binding.
- Angular is built to generally have data flow one-way: top-down
- However when it comes to forms, there are times where it is easier to opt-in to a two-way bind.

Summary

- Forms have a lot of moving pieces, but Angular makes it fairly straightforward.
- FormGroups, FormControls and Validation are the most fundamental tools for Forms in Angular