Building Application using Angular

Forms/Form Group

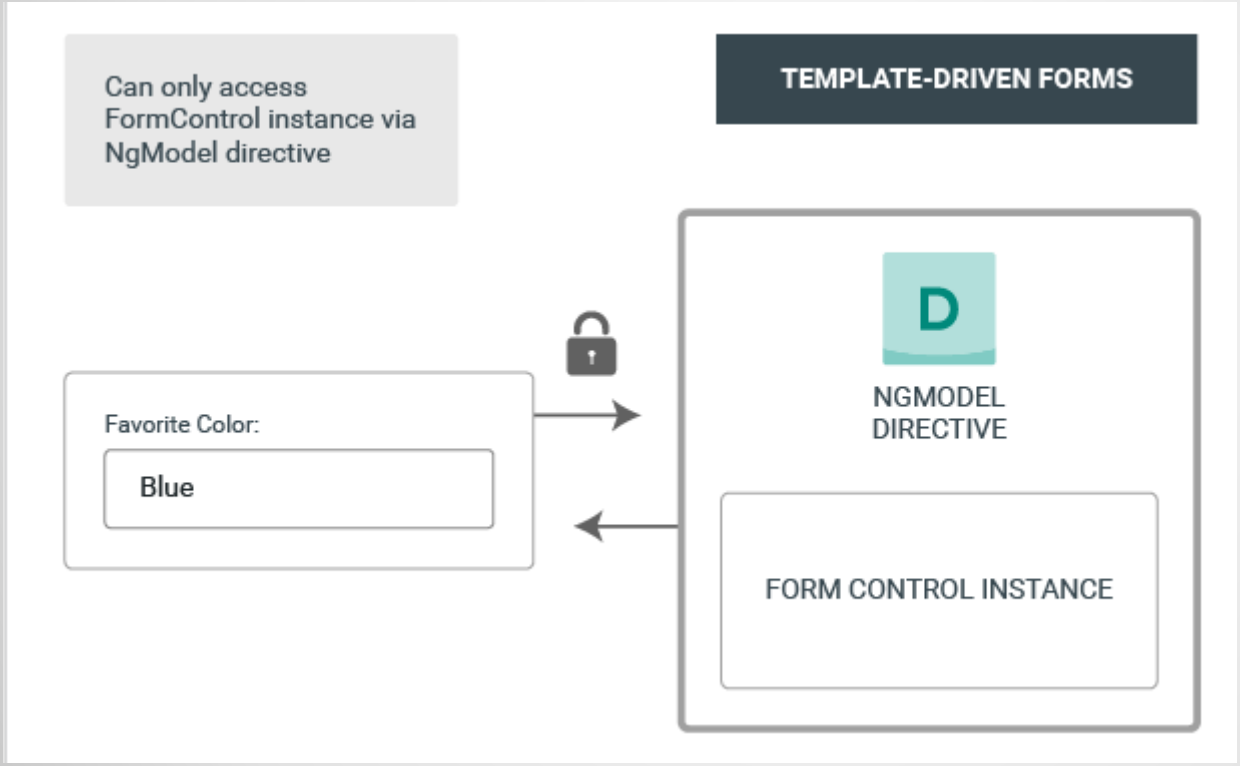
Angular Forms

- As we know many of the application required user forms to enable users to log in, to update a profile, to enter sensitive information, and to perform many other data-entry tasks. For Example:
  - Google Signup
  - Yahoo Signup
  - Login validation
  - Feedback Form

Angular

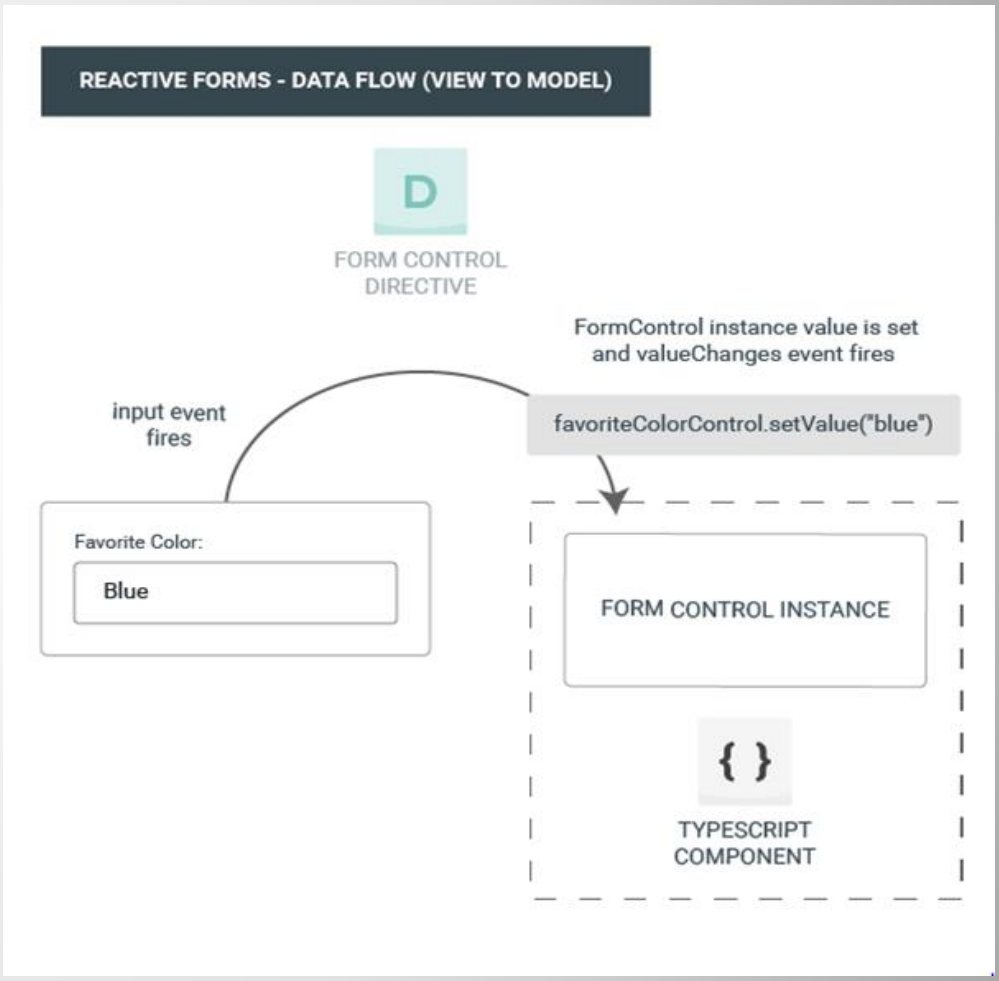
- Angular provides two different approaches to handling user input through forms: reactive and template-driven. Both capture user input events from the view, validate the user input, create a form model and data model to update, and provide a way to track changes
- **Reactive Forms**
  - provide direct, explicit access to the underlying forms object model. Compared to template-driven forms, they are more robust: they're more scalable, reusable, and testable. If forms are a key part of your application, or you're already using reactive patterns for building your application, use reactive forms.
- 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 as email list and sign up form. They are easy to add to an app, but they don’t scale as well as reactive forms. If you have very basic form requirements and logic that can be managed solely in the template, template-driven forms could be good fit.

Reactive Forms

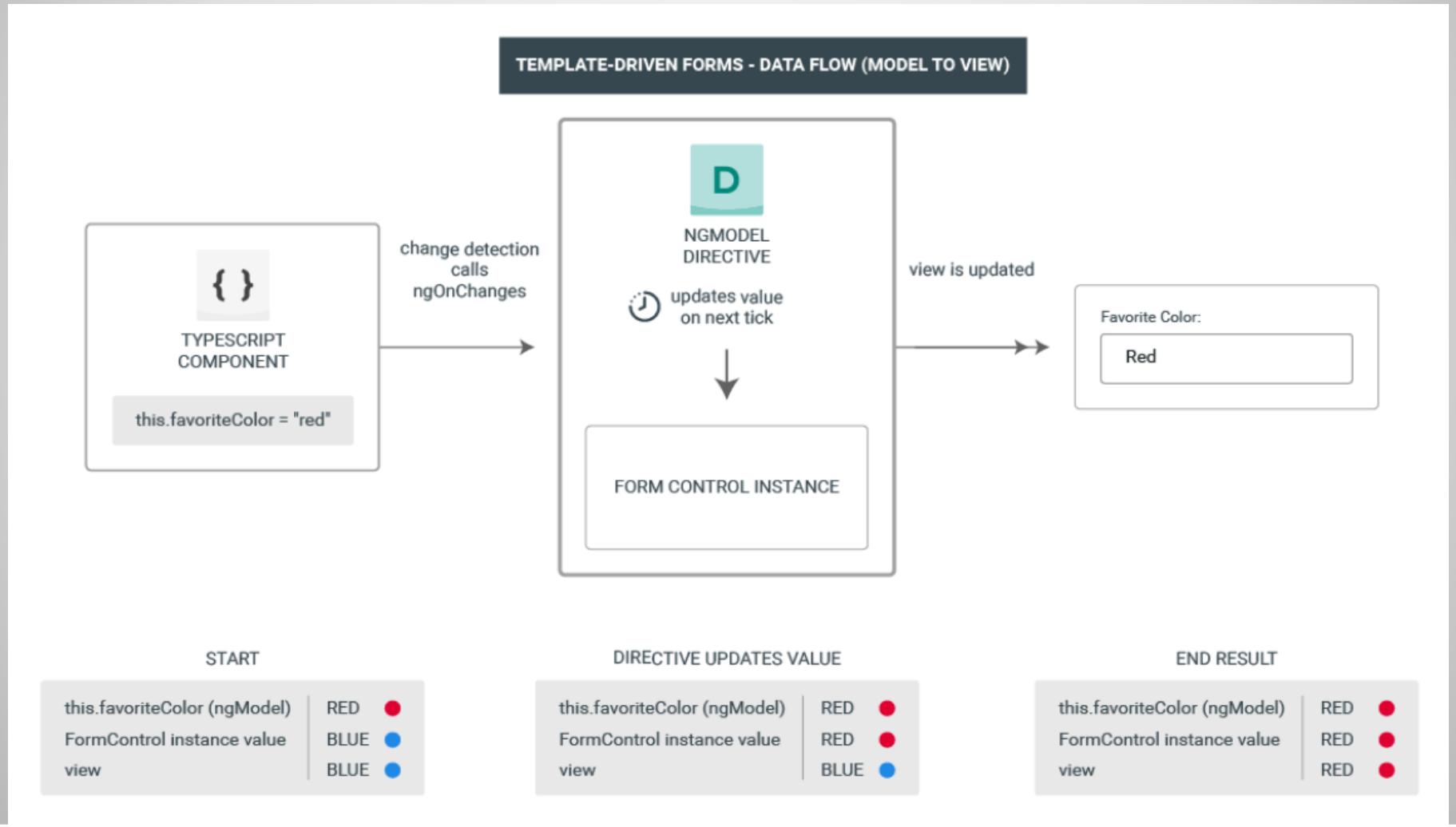


Data Flow in reactive Forms

- The view-to-model diagram shows how data flows when an input field's value is changed from the view through the following steps
  - The user types a value into the input element, in this case the favorite color Blue.
  - The form input element emits an "input" event with the latest value.
  - The control value accessor listening for events on the form input element immediately relays the new value to the FormControl instance.
  - The FormControl instance emits the new value through the valueChanges observable.
  - Any subscribers to the valueChanges observable receive the new value.



Template Driven Form



Mutability of the data model

- The change-tracking method plays a role in the efficiency of your application.
- **Reactive forms** keep the data model pure by providing it as an immutable data structure. Each time a change is triggered on the data model, the FormControl instance returns a new data model rather than updating the existing data model. This gives you the ability to track unique changes to the data model through the control's observable. Change detection is more efficient because it only needs to update on unique changes. Because data updates follow reactive patterns, you can integrate with observable operators to transform data.
- **Template-driven forms** rely on mutability with two-way data binding to update the data model in the component as changes are made in the template. Because there are no unique changes to track on the data model when using two-way data binding, change detection is less efficient at determining when updates are required.

Group of Form Control

- Forms typically contain several related controls. Reactive forms provide two ways of grouping multiple related controls into a single input form.
- Forms typically contain several related controls. Reactive forms provide two ways of grouping multiple related controls into a single input form.
- A form group defines a form with a fixed set of controls that you can manage together. You can also nest form groups to create more complex forms.
- A form array defines a dynamic form, where you can add and remove controls at run time. You can also nest form arrays to create more complex forms.

Steps

- Create a FormGroup instance.
- Associate the FormGroup model and view.
- Save the form data.

```
import { FormGroup, FormControl, FormArray } from '@angular/forms';
  profileForm = new FormGroup({
    firstName: new FormControl(''),
    lastName: new FormControl(''),
  });
<form [formGroup]="profileForm">
  <label>
    First Name:
    <input type="text" formControlName="firstName">
   </label>
    <label>
          Last Name:
          <input type="text" formControlName="lastName">
        </label>
      </form>
```

Form Array

- The FormArray is a way to Manage collection of Form controls in Angular.

```
import { FormGroup, FormControl, FormArray } from '@angular/forms';

hobbies:new FormArray([
     new FormControl()
    ])
get getHobbies() {
    return this.studentForm.get('hobbies') as FormArray;
  }
addHobbies() {
    this.getHobbies.push(new FormControl());
  }

<div formArrayName="hobbies">
<h3>Hobbies</h3> <button (click)="addHobbies()">Add Hobby</button>

<div *ngFor="let hobby of getHobbies.controls; let i=index">
<!-- The repeated hobby template -->
<label>
Hobby:
<input type="text" [formControlName]="i">
</label>
</div>
</div>
```

Importing Form Array from @angular/forms

Creating Form Array Instance

Helps to iterate form array with the help *ngFor

Adding control at run time

Form Builder

- Creating form control instances manually can become repetitive when dealing with multiple forms. The FormBuilder service provides convenient methods for generating controls.
- Use the following steps to take advantage of this service.
- Import the FormBuilder class.

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

- Inject the FormBuilder service.

```
constructor(private fb: FormBuilder) {
this.studentForm=this.fb.group(
      {
      firstName:[''],
      email:[''],
      gender:[''],
      postalCode:''}),
      hobbies:this.fb.array([
      this.fb.control('')
      ])
      }
   );
 }
```

- Generate the form contents.

Validating Form Input

- Form validation is used to ensure that user input is complete and correct.
- You can improve overall data quality by validating user input for accuracy and completeness. This page shows how to validate user input from the UI and display useful validation messages, in both reactive and template-driven forms.
- Use the following steps to add form validation.
  - Import a validator function in your form component.
  - import { Validators } from '@angular/forms';
  - Add the validator to the field in the form.
  - Add logic to handle the validation status.

Example

```
export class FormbuilderComponent{
  studentForm
  PhoneNo:AbstractControl;
 constructor(private fb:FormBuilder)
 {
  this.studentForm=this.fb.group(
    {
      firstName:['',
        Validators.required],
    email:[''],
    gender:['',Validators.required],
    standard:[''],
    PhoneNo:['',Validators.compose([Validators.required,digitValidator])]
 }

  );
  this.PhoneNo=this.studentForm.controls['PhoneNo'];
 }
  }
  //custom validator
   function digitValidator(control: FormControl): { [s: string]: boolean
} {
      if (!control.value.match(/^98/)) {
        return {invalidDigit: true};
      }
    }
```

```
<div class="form-group">
        <label class="control-label col-sm-
2" for="name">First Name:</label>
        <div class="col-sm-10">
            <input type="text" class="form-
control" required placeholder="Enter first name" formCo
ntrolName="firstName">
            <span *ngIf="studentForm.controls.firstName
.valid" >Value required</span>
        </div>
    </div>
```

```
<div>Phone number
    <input type="text" formControlName="PhoneNo">
   <span *ngIf="PhoneNo.invalid"> error </span>
</div>
```

Validation Web Resource

- https://angular.io/api/forms/Validators