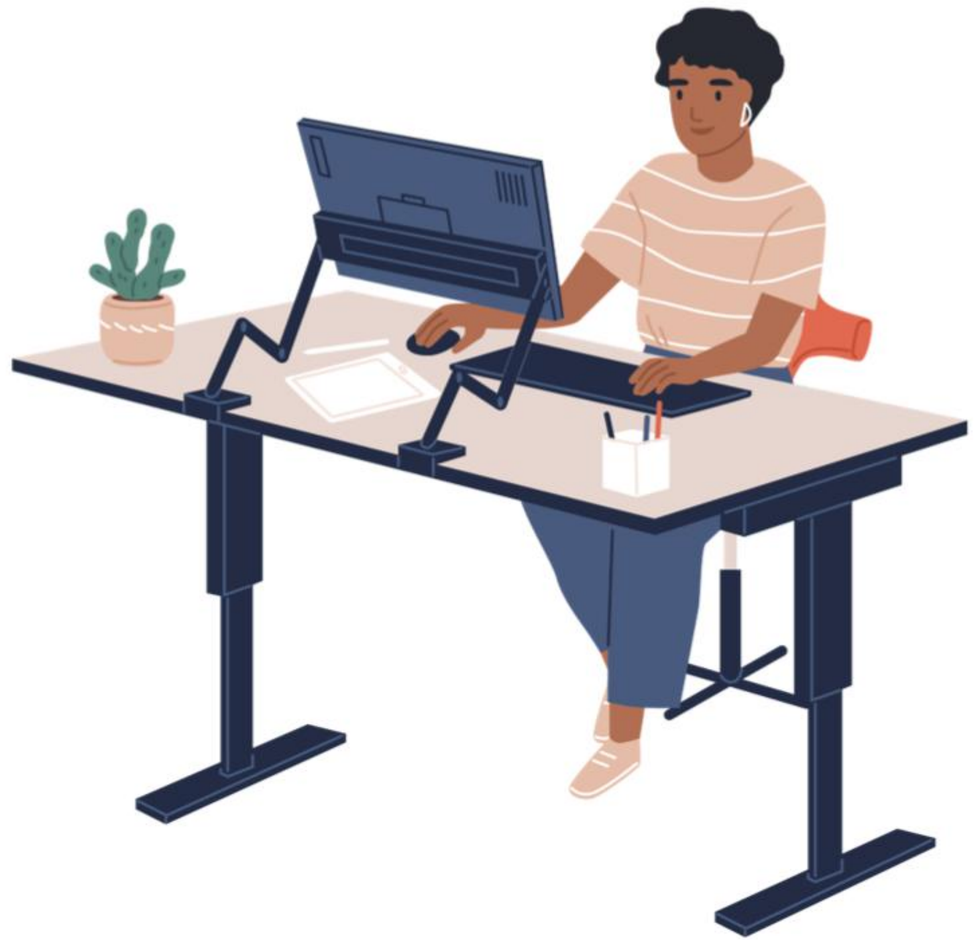
Learning Consolidation

Develop Interactive Reactive Forms Inside SPA



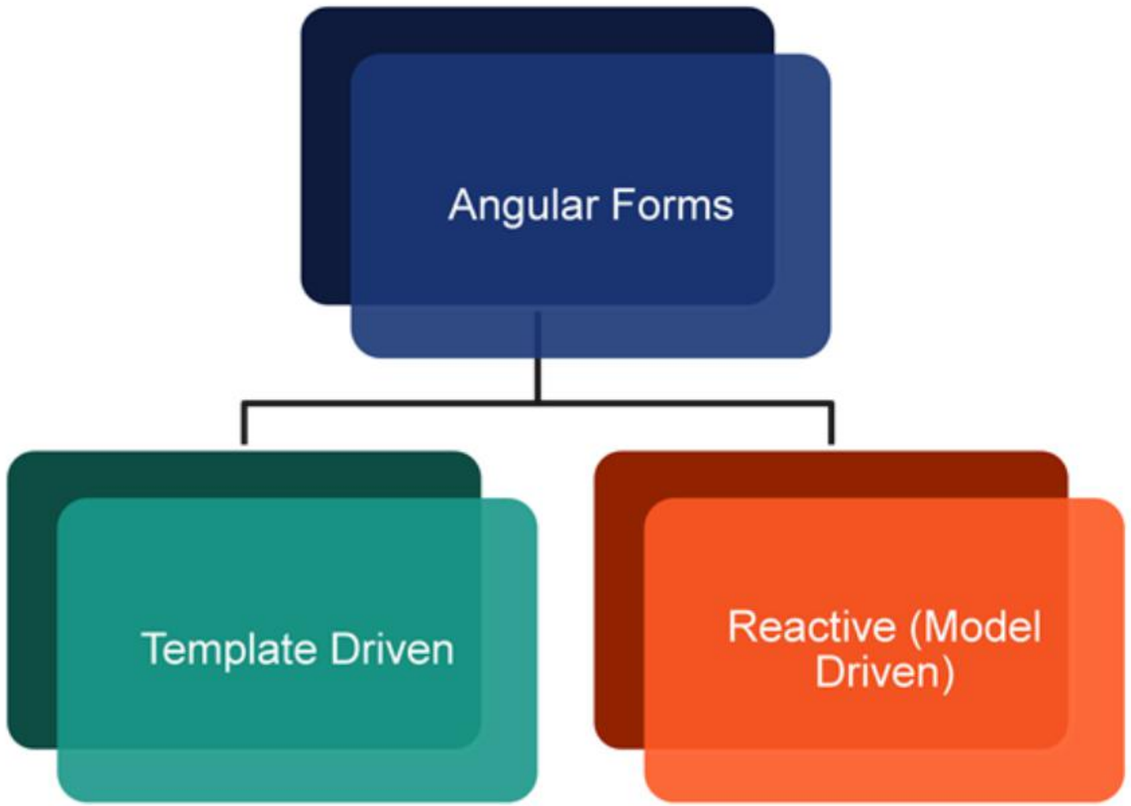
Slide Note

In this Sprint you learned to…

- Explain reactive forms in Angular
- Create, display, and replace FormControl values
- Analyze how data flows in reactive forms
- Contrast grouping form controls using FormGroup and FormArray
- Check the validity of fields in reactive forms using validator functions
- Add custom validators to reactive forms

Slide Note

Different Approaches in Angular Forms



Slide Note

Reactive (Model-Driven) Forms

- Reactive forms are one of the two ways to build forms in Angular.
- These are also called model-driven forms, where the structure of the form is defined in the component class and the HTML content changes depending on the code in the component.
- Reactive forms use an explicit and **immutable** approach to manage the state of a form at a given point in time.
- Each change to the form state returns a new state, which maintains the integrity of the model between changes.
- They are built around observable streams that provide form inputs and values as streams of input values that can be accessed **synchronously**.
- These forms thus provide:
  - synchronous access to the data model
  - immutability with observable operators
  - change tracking through observable streams

Slide Note

• 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 our application, or we're already using reactive patterns for building our application, use reactive forms.

• Reactive forms are more scalable than template-driven forms.

• They provide direct access to the underlying form API, and use synchronous data flow between the view and the data model, which makes creating large-scale forms easier.

• Reactive forms require less setup for testing, and testing does not require deep understanding of change detection to properly test form updates and validation.

Form Foundation Classes

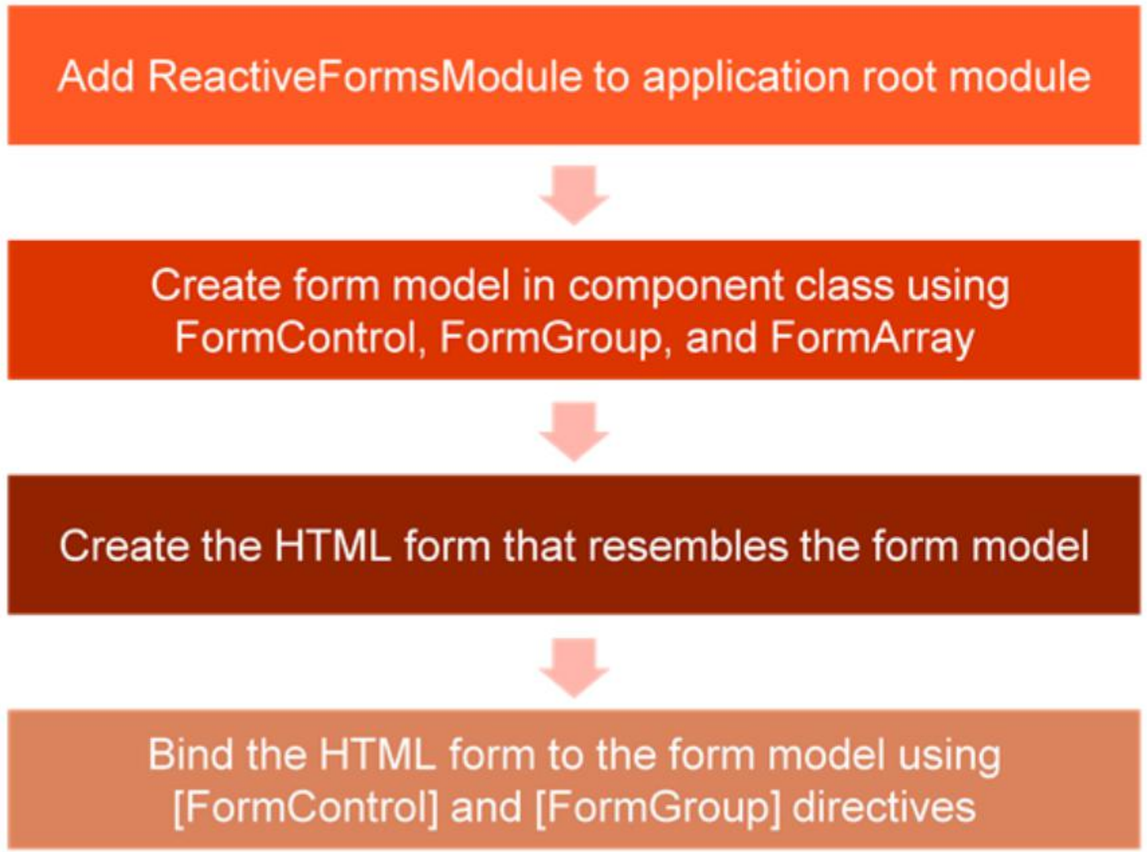
Reactive forms are built on the following base classes.

Base Classes	Details
FormControl	Tracks the value and validation status of an individual form control.
FormGroup	Tracks the value and validation status for a collection of form control instances.
FormArray	Tracks the value and validity state of an array of form control instances. Used for dynamically adding form control elements.
ControlValueAccessor	Creates a bridge between Angular FormControl instances and built-in DOM elements.

Slide Note

Steps For Creating Reactive Forms

Here's how to create reactive forms:



Slide Note

In reactive forms, the form model is the source of truth; it provides the value and status of the form element at any given point in time. This is done through the [formControl] directive on the input element.

Direct access to the form control instance is achieved after the link is created by FormControlDirective.

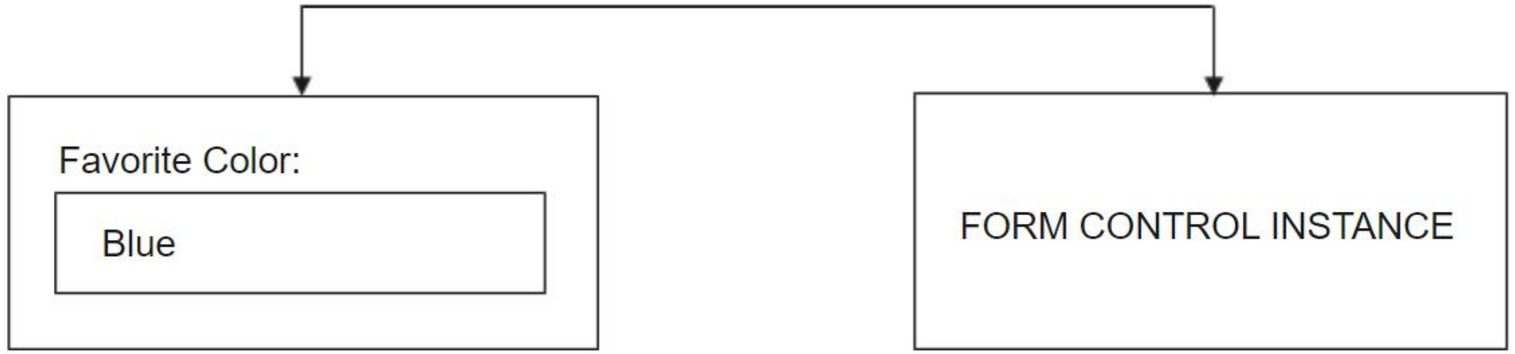
Set up the Form Model

REACTIVE FORMS

D

FORM CONTROL DIRECTIVE

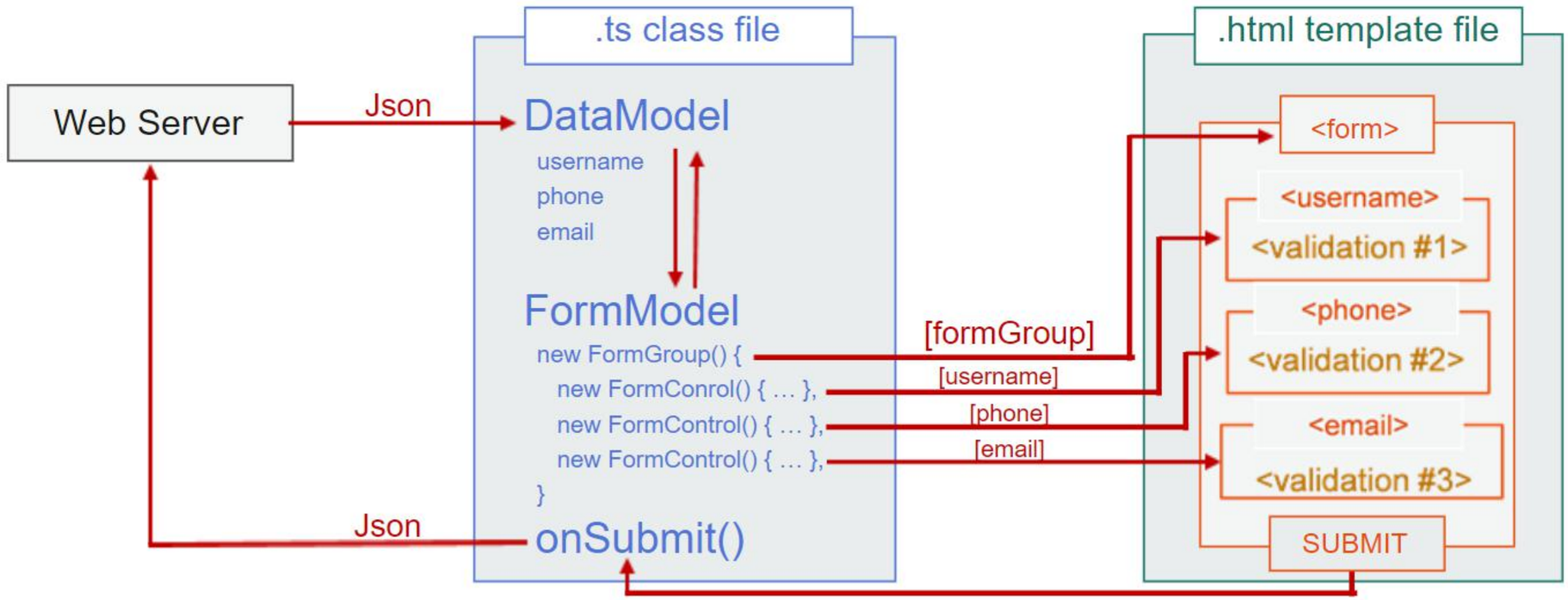
Direct access to `FormControl` instance after link is created by `FormControlDirective`



Slide Note

□ The image shows how the data flows between the component *data model*–which we get from the web server–and a UI-oriented *form model* that retains the states and the values of the HTML form (and its children input elements) presented to the user. This means that you will be able to get in the middle between the data and the form control objects and perform a number of tasks first-hand: push and pull data, detect and react to user changes, implement our own validation logic, perform unit tests, and so on.

Reactive Forms (or Model-Driven Forms)



Slide Note

Mutability of Data in Reactive Forms

- 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 us 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, we can integrate with observable operators to transform data.

Slide Note

Grouping Form Controls

- 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 can be managed together. It is also possible to nest form groups to create more complex forms.
  - A form *array* defines a dynamic form, where controls are added and removed at run time. It is also possible to nest form arrays to create more complex forms.

Slide Note

FormArray is an alternative to FormGroup for managing any number of unnamed controls. As with form group instances, you can dynamically insert and remove controls from form array instances, and the form array instance value and validation status are calculated from its child controls. However, you don't need to define a key for each control by name, so this is a great option if you don't know the number of child values in advance.

To define a dynamic form, take the following steps.

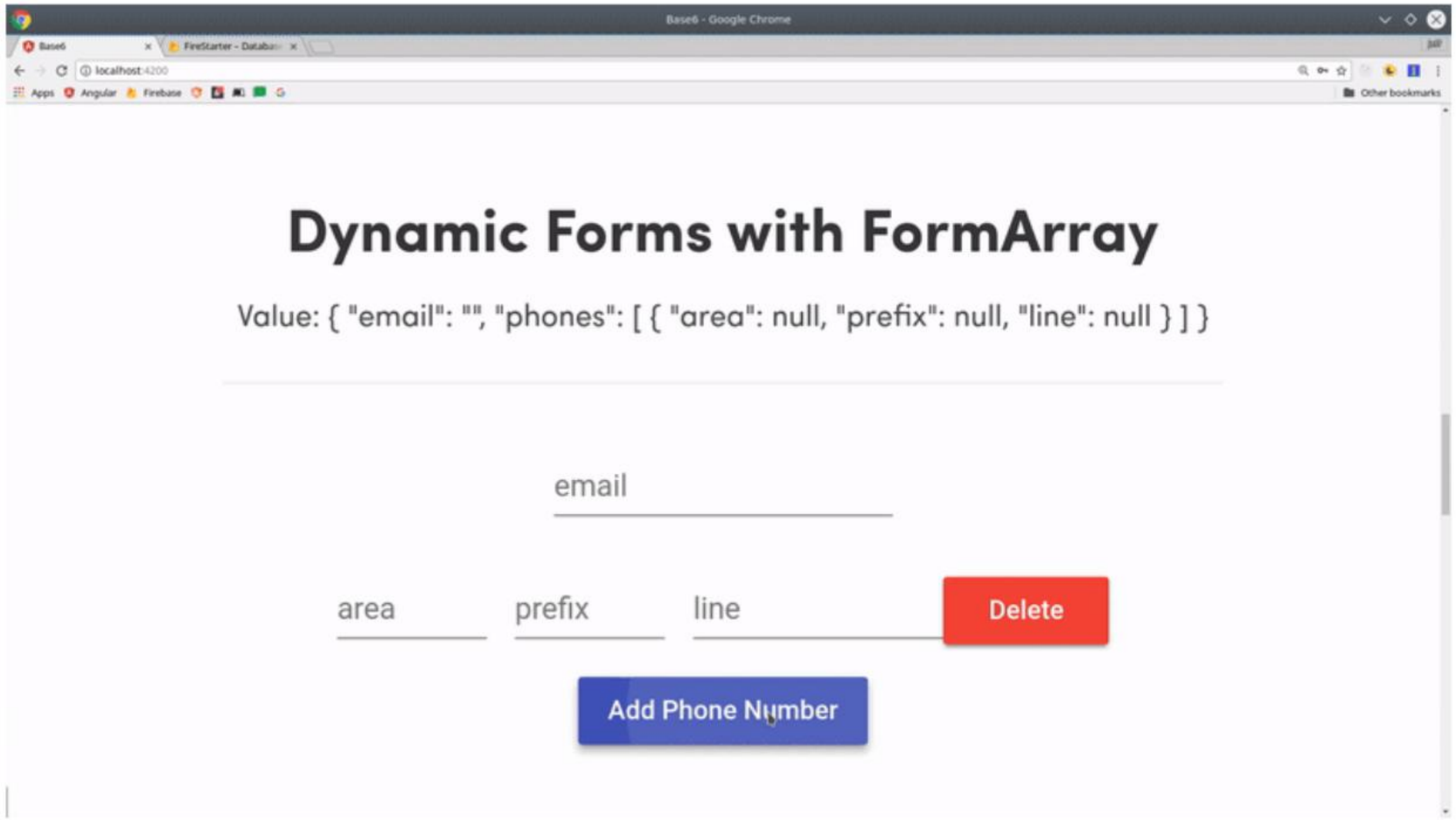
1. Import the FormArray class.

2. Define a FormArray control.

3. Access the FormArray control with a getter method.

4. Display the form array in a template.

Dynamic Forms With FormArray

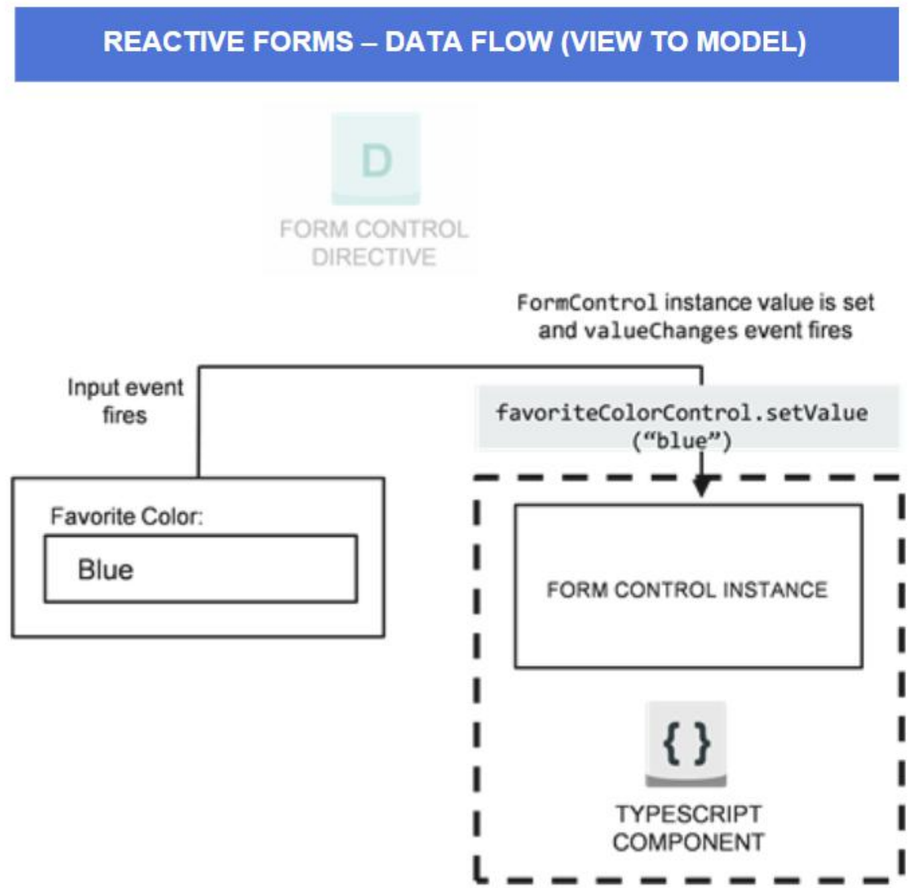


Click on FormArray & for more information FormGroup.
Click here for the reference link.

Slide Note

The view-to-model diagram shows how data flows when an input field's value is changed from the view, which is explained in the next slide.

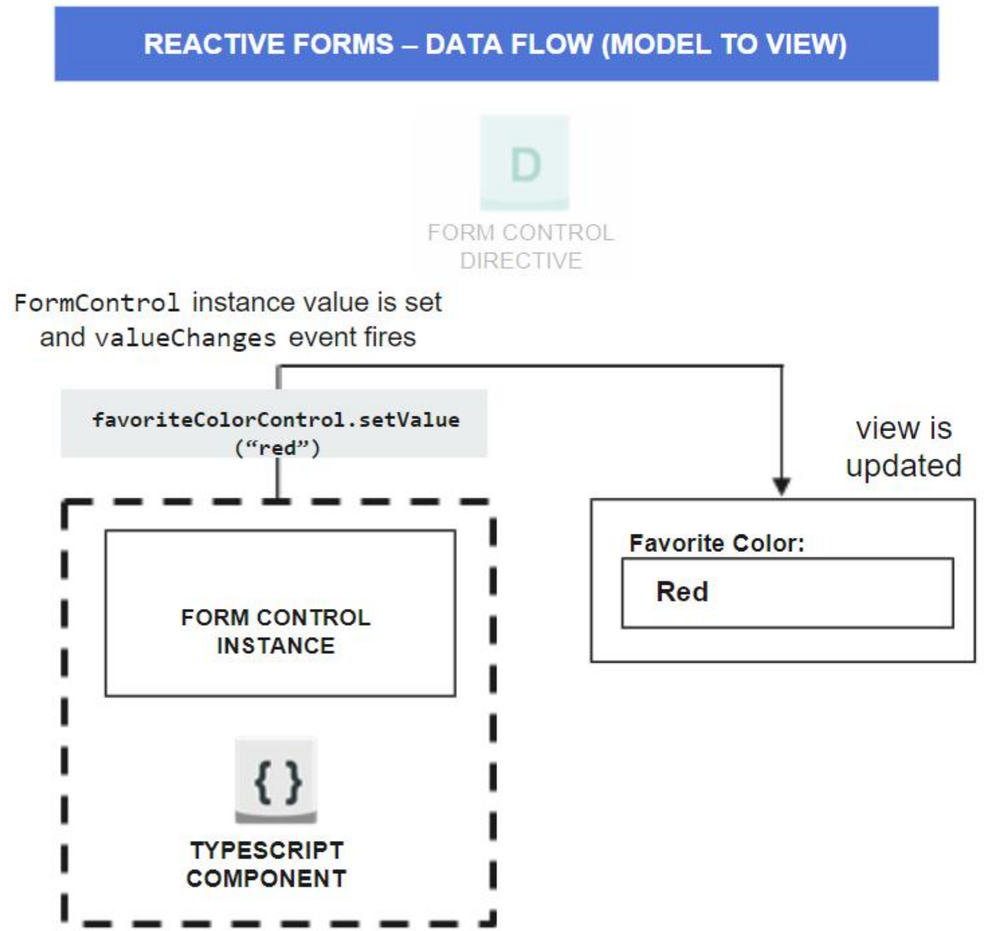
Data Flow From View-to-Model in Reactive Forms



Slide Note

The model-to-view diagram shows how a programmatic change to the model is propagated to the view, which is explained in the next slide.

Data Flow From Model-to-View in Reactive Forms



Slide Note

Form Builder Service

- Manually creating form controls with multiple forms can become repetitive.
- Angular provides FormBuilder service, which offers much easier methods for generating form controls.
- Steps for using FormBuilder service:
    1. Import FormBuilder class.
    2. Inject FormBuilder service by adding it to the component's constructor.
    3. Generate form contents using its factory methods: control(), group() and array().
- In the example, the group() method with the same object is used to define the properties in the model.
- The value for each control name is an array. The initial value of it is given as the first item in the array.

Slide Note

Validating Input Using Functions in Reactive Forms

- In a reactive form, the source of truth is the component class.
- Instead of adding validators through attributes in the template, we add validator functions directly to the form control model in the component class.
- Angular then calls these functions whenever the value of the control changes.
- Validator functions can be either synchronous or asynchronous:
  - **Synchronous validators**: Synchronous functions take a control instance and immediately return either a set of validation errors or null.
    - These are passed as the second argument while instantiating a FormControl.
  - **Asynchronous validators**: Asynchronous functions take a control instance and return a Promise or Observable that later emits a set of validation errors or null.
    - These are passed as the third argument while instantiating a FormControl.

Slide Note

Self-Check

Which of the following modules is used for building reactive forms?

1. FormsModule
2. ReactiveFormsModule
3. DynamicFormsModule
4. FormsInputModule

Slide Note

Self-Check: Solution

Which of the following modules is used for building reactive forms?

1. FormsModule
2. **ReactiveFormsModule**
3. DynamicFormsModule
4. FormsInputModule

Explanation:

- Option 1 is wrong: FormsModule is for Template-driven forms.
- Option 2 & 3 are wrong: There is no such DynamicFormsModule and FormInputModule

Slide Note

Self-Check

When using FormBuilder, how are validation rules specified?

1. As part of the FormControl configuration: heroName: [Validators.required,Validators.maxLength(4)]
2. As part of the FormControl configuration: heroName: (Validators.required,Validators.maxLength(4))
3. As standard HTML validation attributes added to the input element.
4. As part of the input element binding: formControl="[heroName, Validators.required]"

Slide Note

Self-Check: Solution

When using FormBuilder, how are validation rules specified?

1. **As part of the FormControl configuration: heroName: [Validators.required,Validators.maxLength(4)]**

2. As part of the FormControl configuration: heroName: (Validators.required,Validators.maxLength(4))

3. As standard HTML validation attributes added to the input element.

4. As part of the input element binding: formControl="[heroName, Validators.required]"

Explanation:
Option 2 is wrong with respect to syntax.
Option 3 and 4 are wrong since FormBuilder must be used in the Component class and not in the template