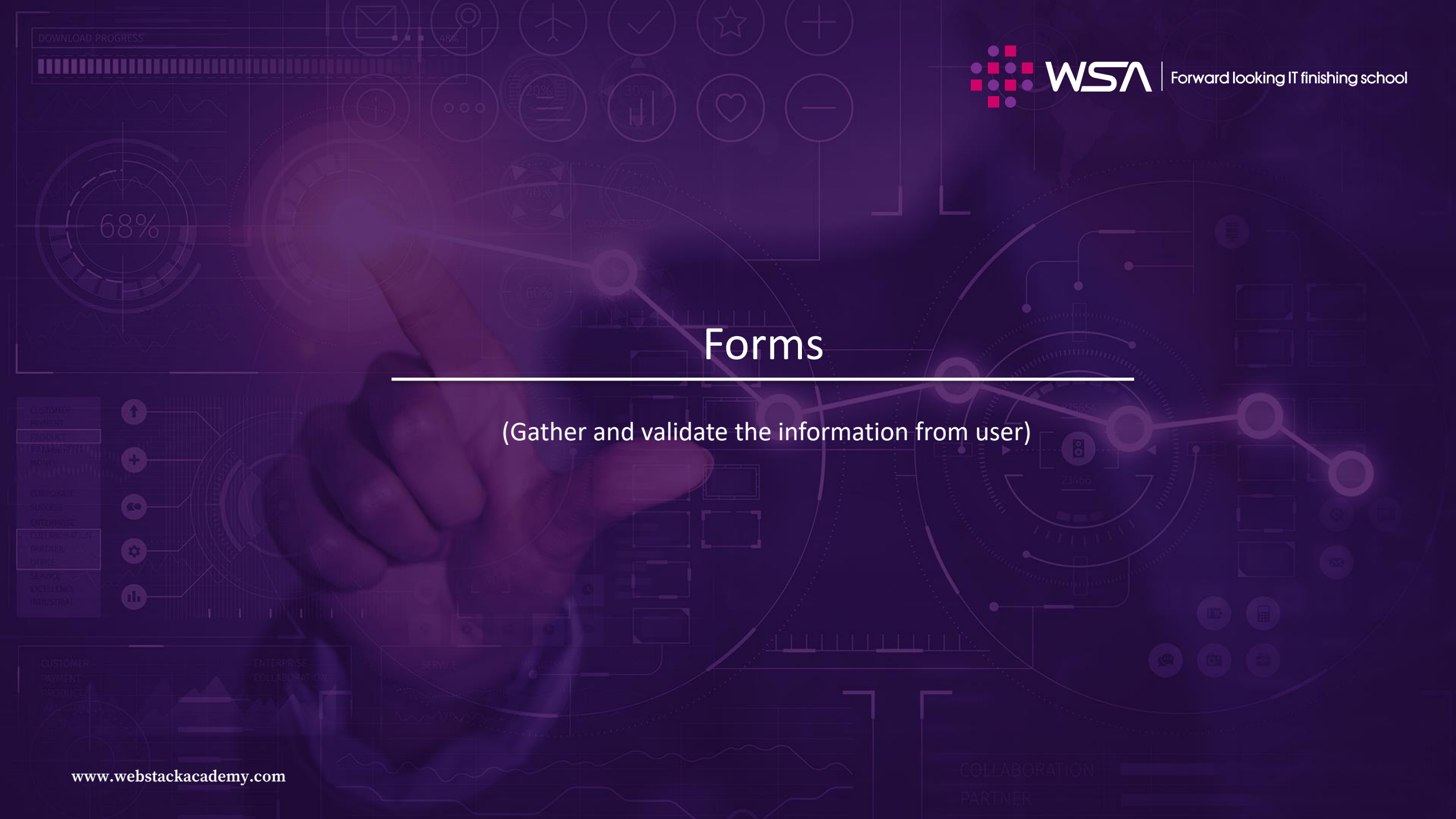
WSA | Forward looking IT finishing school

Forms

Angular



WSA | Forward looking IT finishing school

Forms

(Gather and validate the information from user)

Forms

- Handling user input with forms is the important feature of many common applications.
- Applications use forms to enable users to log in, to update a profile, to enter sensitive information, and to perform many other data-entry tasks.
- 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 and template-driven forms process and manage form data differently. Each offers different advantages.
- **Reactive forms 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** are useful for adding a simple form to an app, such as an email list signup form. They're 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, use template-driven forms.


WSA
Forward looking IT finishing school
Template Driven Forms
(Gather and validate the information from user)
www.webstackacademy.com

Template Driven Forms

- Template driven forms are forms where we write logic, validations, controls etc, in the template part of the code (html code).
- The template is responsible for setting up the form, the validation, control, group etc.
- Suitable for simple forms like login, signup, etc.
- Uses ngModel for reading and writing input-control values.
- Easier to use but Unit testing is challenging in Template driven forms.

Template Driven Forms

Usage Example

Step 1: First, you'll want to make sure that the FormsModule is imported in your app or feature module:

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

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

import { AppComponent } from './app.component';

@NgModule({
  declarations: [ AppComponent ],
  imports: [ BrowserModule,
             FormsModule ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Template Driven Forms

Usage Example

Step 2: We need the ngModel in the form input, and the input must be named too:

```
<input type="text" ngModel name="firstName">
```

- There are cases where we need pass an event listener to the input field, or pass the value of the input to our component, we need assign a template variable to the input to do that.

```
<input type="text" ngModel
       name="firstName"
       #firstName="ngModel"
       (change)="changeLog(firstName)"
>
```

Template Driven Forms

Usage Example

- ngModel is an instance of the FormControl which has quite a number of controls which include dirty, invalid, errors, pristine, touched, untouched, value etc.
- The FormControl class is use to track the state changes of our input.
- By using Form controls, we can easily validate the data.

```
<input type="text" ngModel
       name="firstName"
       #firstName="ngModel"
       (change)="changeLog(firstName)"
       required
>
<div class="alert alert-danger"
     *ngIf="firstName.touched && !firstName.valid">
    FirstName is required
</div>
```

Template Driven Forms

ngForm

- The ngForm is an instance of the FormGroup.
- The FormGroup represents the group of FormControl, each form is a FormGroup because it will have at least one FormControl that gives us access to (ngSubmit) which can be bind to a method in our component.

```
<form #f="ngForm" (ngSubmit)="submit(f)">
    <input type="text" ngModel
           name="firstName"
           #firstName="ngModel"
           (change)="“firstNameLog(firstName)"
    >
    <input type="submit" value="submit">
</form>
```

Template Driven Forms

ngModelGroup

- At times, when building a complex form, need might arise where we need make a particular object a parent to some other inputs, so we can access those inputs under the parent.
- Hence the need for ngModelGroup. We can access the firstName and lastName under the person object.

```
<form #f="ngForm" (ngSubmit)="submit(f)">
    <div ngModelGroup="person">
       <input type="text" ngModel
              name="firstName"
              #firstName="ngModel"
              (change)="firstNameLog(firstName)"
       >
       <input type="text" ngModel
              name="lastName"
              #lastName="ngModel"
              (change)="lastNameLog(lastName)"
       >
    </div>
    <input type="submit" value="Submit">
  </form>
```

Reactive Forms

(Gather and validate the information in Component)

Reactive Forms

- Angular reactive forms facilitate a reactive style of programming to get data in and out of the form from where it is been defined in the component to the template visa versa through the use of Form Model and Form Directives.
- Reactive forms offer the ease of using reactive patterns, testing, and validation.
- Reactive forms are forms where we write logic, validations, controls in the components class part of the code unlike the template driven forms where control is done in the template.
- The reactive form is flexible and can be use to handle any complex form scenarios.
- We write more component code and less html code which make unit testing easier.

Reactive Forms

Usage Example

Step 1: To use reactive form, we need to explicitly FormsModule, ReactiveFormsModule in app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { FormsModule,ReactiveFormsModule } from '@angular/forms';

import { AppComponent } from './app.component';

@NgModule({
  declarations: [ AppComponent ],
  imports: [ BrowserModule,
             FormsModule,ReactiveFormsModule
           ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Reactive Forms

FormControl and FormGroup

- The FormControl is a class that powers an individual form control, tracks the value and validation status, whilst offering a wide set of public API methods. Below is a basic example of a FormControl

```
'firstname': new FormControl('')
```

- Form Control has a constructor which is having in the below form

```
constructor(formState?: any, validators?: ValidatorFn |
AbstractControlOptions | ValidatorFn[ ], asyncValidator?:
AsyncValidatorFn | AsyncValidatorFn[ ])
```

Initial Value

Validation on the field, e.g required,minlength, maxlength etc.

Reactive Forms

FormControl and FormGroup

- The FormGroup is a group of FormControl instances, keeps track of the value and validation status for the said group, and also offers public APIs. Below is a basic example of theFormGroup

```
myGroup = new FormGroup({
    'firstname': new FormControl(''),
    'password': new FormControl('')
  });
```

- Form Control has a constructor which is having in the below form

```
constructor(controls: { [key: string]: AbstractControl; },
validatorOrOpts?: ValidatorFn | AbstractControlOptions | ValidatorFn[],
asyncValidator?: AsyncValidatorFn | AsyncValidatorFn[])
```

A collection of child controls

Validation on form

Asynchronous Validation

Reactive Forms

Validations

- To add validation to a reactive form, we need to import the Validators class from @angular/forms and pass the validation(s) in as a second argument to our FormControl instances.

```
import { ReactiveFormsModule, FormsModule, FormGroup, FormControl, Validators}
                                                        from '@angular/forms';

myGroup = new FormGroup({
    'firstname': new FormControl('',Validators.required),
    'password': new FormControl('',Validators.required)
});
```

Reactive Forms

Custom Validations

- Lets say, we want to validate that username should not have any space. Angular have only limited built-in validation which cannot be used to validate the before mentioned condition.
- To implement our own validation, we can create custom validation of our own.
- In Angular, creating a custom validator is as simple as creating another function inside a class.
- Create a ts file which should have the extension of **filename.validators.ts**
- The only thing you need to keep in mind is that validator function takes one input parameter of type AbstractControl and it returns an object of key value pair if the validation fails.

```
import { AbstractControl, ValidationErrors } from "@angular/forms";

export class UsernameValidators {
  static cannotContainSpace(control: AbstractControl):
   {[key: string]: Boolean} | null {
        return null;
    }
}
```

Reactive Forms

Custom Validations

- The type of the first parameter is AbstractControl because it is a base class of FormControl, FormArray, and FormGroup, and it allows you to read the value of the control passed to the custom validator function. The custom validator returns either of the following:
  - If the validation fails, it returns an object, which contains a key value pair. Key is the name of the error and the value is always Boolean true.
  - If the validation does not fail, it returns null.

```
import { AbstractControl, ValidationErrors } from "@angular/forms";

export class UsernameValidators {
    static cannotContainSpace(control:AbstractControl) : ValidationErrors | null {
        if((control.value as string).indexOf(' ')>=0)
            return {cannotContainSpace:true};
        return null;
    }
}
```

WSA | Forward looking IT finishing school

WebStack Academy

#83, Farah Towers,
1st Floor, MG Road,
Bangalore – 560001

M: +91-809 555 7332
E: training@webstackacademy.com

WSA in Social Media: