Angular Hands-on Workshop

Ervin Suhanko
Front End Software Engineer,
Team Lead & Mentor

Prerequisite

- Basic understanding of HTML, CSS, and JavaScript
- Basic understanding of Programming
- Familiarity with ES6 and Typescript is helpful

5th – 6th – 7th – 8th – 9th Sep'22 | 9:30 – 10:30 AM EST

Agenda

Day 1	Day 2	Day 3	Day 4	Day 5
Angular Introduction and Framework Overview	Modules	Routing	RxJS Essentials	Tips, Tricks and Best Practices
Angular CLI (Command-line Interface tool)	Components and Data flow	Pipes and Directives	Services and HTTP	
Angular Project Structure		Reactive Forms		
TypeScript Essentials				

Routing

- Routes are defined in a route definition table that in its simplest form contains a path and component reference
- Components are loaded into the router-outlet directive
- We can navigate to routes using the routerLink directive
- The router uses history.pushState which means we need to set a base-ref tag to our index.html file

Routing

Why to use it?

Routing help us to directing users to different pages on the option they choose on the main page.

Routing

- Lazy Loading
- Preloading Modules
- Router Events
- Guards

Demo

Routes

Pipes

- Transform bound properties before display
- Runs on EVERY change detection cycle
- Built-in pipes: date, number, decimal, percent, currency, json, etc.
- Custom pipes

Demo

Pipes

Directives

- A directive is a class decorated with @Directive
- Tree types of Directives: structural directives attribute directives and Components
- A component is just a directive with added template features

Directives

Structural Directives

- A structural directive changes the DOM layout by adding and removing DOM elements.
- Asterisks indicate a directive that modifies the HTML
- It is syntactic sugar to avoid having to use template elements directly
- *ngIf, *ngFor, ngSwitch (*ngSwitchCase, *ngSwitchDefault)
- ngClass, ngStyle, ngModel

Directives

Attribute Directives

Change the **appearance** or **behavior** of DOM elements and Angular components with attribute directives.

```
import { Directive, ElementRef } from '@angular/core';

@Directive({
  selector: '[appHighlight]'
})
export class HighlightDirective {
    constructor(private el: ElementRef) {
        this.el.nativeElement.style.backgroundColor = 'yellow';
    }
}
```

Demo

Directives

Forms

Angular provides two different approaches to handling user input through forms:

- reactive forms
- template-driven forms.

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 and template-driven forms process and manage form data differently.

Forms

Reactive vs Template-driven forms

	REACTIVE	TEMPLATE-DRIVEN
Setup of form model	Explicit, created in component class	Implicit, created by directives
Data model	Structured and immutable	Unstructured and mutable
Data flow	Synchronous	Asynchronous
Form validation	Functions	Directives

Reactive Forms

Angular reactive forms follow a model-driven approach to handle form input whose values can be changed over time. These are also known as model-driven forms. In reactive forms, you can create and update a simple form control, use multiple controls in a group, validate form values, and implement more advanced forms.

Reactive forms use an explicit and immutable approach to manage the state of the form at a given point of time. When we change the form state, it returns a new state which manages the integrity of the models between changes. In reactive forms, you build your own representation of a form in the component class.

Reactive Forms

Adding a basic form control

Step 1: Register the reactive forms module

```
import { ReactiveFormsModule } from '@angular/forms';

@NgModule({
  imports: [
    // other imports ...
    ReactiveFormsModule
  ],
})
export class AppModule { }
```

Reactive Forms

Adding a basic form control

Step 2: Generate a new FormControl

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

@Component({
  selector: 'app-name-editor',
  templateUrl: './name-editor.component.html',
  styleUrls: ['./name-editor.component.css']
})
export class NameEditorComponent {
  name = new FormControl('');
}
```

Reactive Forms

Adding a basic form control

Step 3: Register the control in the template

```
<label for="name">Name: </label>
<input id="name" type="text" [formControl]="name">
```

Reactive Forms

Grouping Form controls

Form Group defines a form with a fixed set of controls that you can manage together:

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

@Component({
  selector: 'app-profile-editor',
  templateUrl: './profile-editor.component.html',
  styleUrls: ['./profile-editor.component.css']
})
export class ProfileEditorComponent {
  profileForm = new FormGroup({
    firstName: new FormControl(''),
    lastName: new FormControl(''),
  });
}
```

Reactive Forms

Grouping Form controls

A form group tracks the status and changes for each of its controls, so if one of the controls changes, the parent control also emits a new status or value change

```
<form [formGroup]="profileForm">

  <label for="first-name">First Name: </label>
  <input id="first-name" type="text" formControlName="firstName">

  <label for="last-name">Last Name: </label>
  <input id="last-name" type="text" formControlName="lastName">

</form>
```

Reactive Forms

Save form data

The FormGroup directive listens for the submit event emitted by the form element and emits an **ngSubmit** event that you can bind to a callback function. Add an ngSubmit event listener to the form tag with the onSubmit() callback method.

```
<form [formGroup]="profileForm" (ngSubmit)="onSubmit()">
```

```
onSubmit() {
  // TODO: Use EventEmitter with form value
  console.warn(this.profileForm.value);
}
```

Reactive Forms

- Nested Form Groups
- Update parts of the data model

  setValue(); patchValue();
- Form Builder

  constructor(private fb: FormBuilder) { }
- Validating form input

Demo

Reactive Forms