Section 2: The Basics: Table of Contents

[15. How an Angular App gets Loaded and Started 2](#_Toc64552847)

[16. Components are Important! 2](#_Toc64552848)

[17. Creating a New Component 2](#_Toc64552849)

[18. Understanding the Role of AppModule and Component Declaration 3](#_Toc64552850)

[20. Creating Components with the CLI & Nesting Components 3](#_Toc64552851)

[21. Working with Component Templates 4](#_Toc64552852)

[22. Working with Component Styles 4](#_Toc64552853)

[23. Fully Understanding the Component Selector 5](#_Toc64552854)

[25. What is Databinding? 5](#_Toc64552855)

[26. String Interpolation 5](#_Toc64552856)

[27. Property Binding 6](#_Toc64552857)

[28. Property-Binding vs String Interpolation 6](#_Toc64552858)

[29. Event Binding 7](#_Toc64552859)

[30. Bindable Properties and Events 7](#_Toc64552860)

[31. Passing and Using Data with Event Binding 7](#_Toc64552861)

[32. Important: FormsModule is Required for Two-Way-Binding! 8](#_Toc64552862)

[33. Two-Way-Binding 8](#_Toc64552863)

[36. Understanding Directives 8](#_Toc64552864)

[37. Using ngIf to Output Data Conditionally 9](#_Toc64552865)

[38. Enhancing ngIf with an Else Condition 9](#_Toc64552866)

[39. Styling Elements Dynamically with ngStyle 9](#_Toc64552867)

[40. Applying CSS Classes Dynamically with ngClass 10](#_Toc64552868)

[41. Outputting Lists with ngFor 10](#_Toc64552869)

[43. Getting the Index when using ngFor 10](#_Toc64552870)

# 15. How an Angular App gets Loaded and Started

Angular gets started with the main.ts file, which through bootstrapModule starts our angular application by passing an appModule to the method. This refers to app.module.ts, where we see that the bootstrap array lists all of the modules that should be known to angular upon running the index.html file. This includes the AppComponent, which is setup in the app.component.ts file.

# 16. Components are Important!

In Angular, we typically start with the AppComponent, which is the root component. We will nest other custom components to the html template off of the root component.

# 17. Creating a New Component

The @Component decorate needs a javascript object passed to it for configuration. Here, we setup metadata for the class that tells angular what to do with the class.

1. selector - used to setup HTML tag for which to reference in other components. The selector name needs to be unique and not overwrite a default HTML element. You typically prefix it with app-

2. templateUrl - reference a user-created component.html with a relative path

Decorator - a TypeScript feature which allow you to enhance your classes or other elements used in your code.

In a new component.ts file, you will add the @Component  decorator. Decorators always have an @ prefix. However, before using the Component decorator, it has to be imported from the angular/core library with import { component } from '@angular/core';

The root app component is special in that it is the only app component that is bootstrapped in the app module. Selectors for other components will be added in the app.component.html file.

# 18. Understanding the Role of AppModule and Component Declaration

Before using our own component, we need to edit the app module. By default, Angular will not scan all of the files in the project.

While Angular uses components to build web pages, it uses modules to bundle different functional pieces into packages. It tells Angular what features does the app have and use.

The app module is built by using the @NgModule decorator imported from @angular/core. We register new components in the declarations array first off. However, you also need to import your new component from the correct folder as done with the app component. This needs to be done so TypeScript knows where the component is at, it's a TS feature.

# 20. Creating Components with the CLI & Nesting Components

Components can be created with the CLI as opposed to by hand with either of the following commands:

ng generate component servers

ng g c servers

The CLI should automatically update the app module, but you should verify that it was edited properly with the new component. Components can be nested in others as often as needed (i.e.- two server component tags nested in a servers component html)

# 21. Working with Component Templates

In the Component decorator, you absolutely must have either the template or templateUrl property included. templateUrl enables you to reference another local html file, while template lets you input html tags directly into a string. However, text wrapping is not enables when doing string input. Multi-line strings with line wrapping will work if using backticks `<tag></tag>` instead of single quotes '<tag></tag>' .

If you have more than 3 lines of html code for your template, you should use an external html file. Otherwise, just using the TS file is okay.

# 22. Working with Component Styles

In the component css file, you can write some standard css to for example, overwrite the style of the h3 tag. Ex:

1. h3 {
2. color: darkblue;
3. }

However, you can also input shortened amounts of css code directly to the styles property instead of using the styleUrls property of the app component TS file. styles (inline) and styleUrls (external) are in array form because you can use them to reference multiple style sheets if desired.

For styling within the html file, you can add bootstrap-provided css classes with the div tags. For example:

1. <div class="container">
2. <div class="row">
3. <div class="col-xs-12">
4. </div>
5. </div>
6. </div>

# 23. Fully Understanding the Component Selector

You can also select by class with your selector. In the component TS file, you would use a period before the selector name to make it a class. However, selecting by id will not work. Overall, the element style is typically used for components.  
Ex: selector: '.app-servers',

In the referencing html file, you would reference it like a div class:  
<div class="app-servers"></div>

The component selectors we create work the same way the CSS selectors work for styling tags. Therefore, you aren't limited to selecting by element. You can use square brackets around your selector name in the component TS file to make it an attribute (ex: selector: '[app-servers]',), which will break the code.

You would then have to go to the referencing html file and reference the selector like an attribute: <div app-servers></div>

# 25. What is Databinding?

Databinding is communication between TypeScript code and the HTML template. You can output data to the template through methods such as string interpolation   
({‌{ data }}) or property binding ([property] = "data").

Event Binding ((event) = "expression") lets you bind click-events, for example, to trigger code from the html via a user-event. Two-Way-Binding ([(ngmodel)] = "data") is a combination of output data and reactions to user events.

# 26. String Interpolation

String interpolation works by transmitting a property in the TS file to the HTML template by enclosing the property name in double curly braces in the HTML.

Rules for String Interpolation:

1. Any property or method included in curly braces has to return a string.

2. You cannot write multi-line expressions (block expressions).

For properties with a number typing, they will be easily converted to strings.

# 27. Property Binding

Essentially any HTML element properties can be bound to. However, certain properties will require certain types when binding. For instance, the disabled property requires a boolean value.

You can also bind to other properties, such as of directives and custom components.

As an example of data binding, we run the following code the TS file to initiate a property that would allow a button to work after a set timeout of 2 seconds:

1. allowNewServer = false;
3. constructor() {
4. setTimeout(() => {
5. this.allowNewServer = true;
6. }, 2000)
7. }

To bind, we enclose the disabled attribute in the HTML with square brackets, which lets angular know that we want to bind some property to the attribute. The expression portion indicates which property to bind with. The exclamation point returns the opposite of a boolean.

[disabled]="!allowNewServer"

You can add a button in the html using the bootstrap CSS classes, which aren't related to angular:  
<button class="btn btn-primary">Add Server</button>

The button can have a disabled attribute added to disable it after the class declaration.

# 28. Property-Binding vs String Interpolation

String Interpolation vs. Property Binding:

If you want to output something to your template and print text to it, use string interpolation. But if you want to dynamically change some property, use property binding.

The curly braces intended for string interpolation will not working within an expression used for property binding, as they will break it.

# 29. Event Binding

Like with property binding, we are gonna use a method to be triggered upon a user event in the TS file. The word 'on' is generally used when dealing with user events.

1. onCreateServer() {
2. this.serverCreationStatus = 'Server was created!';
3. }

# 30. Bindable Properties and Events

How do you know to which Properties or Events of HTML Elements you may bind? You can basically bind to all Properties and Events - a good idea is to console.log()  the element you're interested in to see which properties and events it offers.

**Important**: For events, you don't bind to onclick but only to click (=> (click)).

The MDN (Mozilla Developer Network) offers nice lists of all properties and events of the element you're interested in. Googling for YOUR\_ELEMENT properties  or YOUR\_ELEMENT events  should yield nice results.

# 31. Passing and Using Data with Event Binding

Say we want to output what the user entered in the input. We can do this by passing $event, a reserved variable name when event binding. The click and input events ship some data when fired, which can be passed by using the $event passed as an argument anywhere in the expression.

1. <input
2. type="text"
3. class=form-control
4. (input)="onUpdateServerName($event)">

We pass event into the method as type Event and use inspect after logging it to the console to determine where the input value is stored.

1. onUpdateServerName(event: Event) {
2. console.log(event);
3. this.serverName = (<HTMLInputElement>event.target).value;
4. }

The IDE doesn't like event.target since it is an input element, so it needs to be specifically cast with parenthesis and carets as an HTMLInputElement.

# 32. Important: FormsModule is Required for Two-Way-Binding!

Important: For Two-Way-Binding (covered in the next lecture) to work, you need to enable the ngModel  directive. This is done by adding the FormsModule  to the imports[]  array in the AppModule.

You then also need to add the import from @angular/forms  in the app.module.ts file:

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

# 33. Two-Way-Binding

Two-way data-binding combines properties and databinding. We use square bracket and parenthesis around ngModel and set it equal to the property name. Ex:  
[(ngModel)]="serverName"

This will trigger on input events to update the property name, but it will also update the value in the input element if we change the property somewhere else.

# 36. Understanding Directives

Directives are instructions in the DOM. For example, using the selector as an instruction to add a component for Angular. We typically add directives with the attribute selector, but they can be configured just like the component with CSS classes or the element style. Ex:

<p appTurnGreen>Receives a green background!</p>

Angular would find this directive with the directive decorator:

1. @Directive({
2. selector: '[appTurnGreen]'
3. })
4. export class TurnGreenDirective {
5. ...
6. }

# 37. Using ngIf to Output Data Conditionally

ngIf is added by using a star, which is needed since ngIf is a structural directive that either adds the element or doesn't add the element. We set up a condition between the quotation marks, which has to be any expression that can return a true or false.

<p \*ngIf="binaryExpression">Text here</p>

It's important to distinguish that the content is added or removed from the DOM based on the expression result. It is not there all of the time.

# 38. Enhancing ngIf with an Else Condition

An else condition for ngIf can be achieved by using the ng-template tag with a local reference marker (#). The ng-template is a component/directive shipping with angular with which can you mark places in the DOM. With the local reference, you can then include an else condition in the expression for ngIf

1. <p \*ngIf="binaryExpression; else markerName">Text here</p>
2. <ng-template #markerName>
3. <p>Alternate Text here</p>
4. </ng-template>

# 39. Styling Elements Dynamically with ngStyle

An else condition for ngIf can be achieved by using the ng-template tag with a local reference marker (#). The ng-template is a component/directive shipping with angular with which can you mark places in the DOM. With the local reference, you can then include an else condition in the expression for ngIf

1. <p \*ngIf="binaryExpression; else markerName">Text here</p>
2. <ng-template #markerName>
3. <p>Alternate Text here</p>
4. </ng-template>

# 40. Applying CSS Classes Dynamically with ngClass

Similar to ngStyle, ngClass can be added as another directive with a self-named property that uses property binding. The ngClass property expects a javascript object with key-value pairs of the class name and conditional expression. The key name should be a CSS class that uses camelCase or hyphenation paired with single quotation marks.

In ts file (or CSS file):

1. @Component:({
2. styles: [`
3. .online {
4. color: white;
5. }
6. `]

In html file:

<p [ngClass]="{className: binaryExpression }">Text</p>

# 41. Outputting Lists with ngFor

A structural directive that can be used is the \*ngFor directive. The ngFor directive will iterate the html component it is inserted in for an assigned single dynamic variable instance of each value in an array variable specified. In the example below, server is assigned each value from the array, servers.

<app-server \*ngFor="let server of servers"></app-server>

# 43. Getting the Index when using ngFor

You can use a semicolon after the expression for ngFor directive in order to create an index variable for the loop that can be used for other purposes such as styling.

\*ngFor="let logItem of log; let i = index"