To change port to run on - ng serve --port 4401; in this case, port 4401 is used

Ng version – Gets angular version

**C:\>cd \AngularProjects**

**C:\AngularProjects>ng new AzureRouting --routing false --style css --skip-git --skip-tests**

ng new todo --routing false --style css --skip-git --skip-tests

ng new mattabs --routing false --style css --skip-git --skip-tests

ng new todo --routing false --style css --skip-git --skip-tests

ng new databindings --routing false --style css --skip-git --skip-tests

ng new reactiveextensions --routing false --style css --skip-git --skip-tests

ng new angularprojectstools --routing false --style css --skip-git --skip-tests

ng new exampleproject --routing false --style css --skip-git --skip-tests

ng new builtindirectives --routing false --style css --skip-git --skip-tests

ng new eventsforms --routing false --style css --skip-git --skip-tests

ng new structuraldirectives --routing false --style css --skip-git --skip-tests

ng new attributedirectives --routing false --style css --skip-git --skip-tests

ng new httprequests --routing false --style css --skip-git --skip-tests

ng new JavaScriptPrimer --routing false --style css --skip-git --skip-tests

ng new AzureFunctionTests --routing false --style css --skip-git --skip-tests

ng new components --routing false --style css --skip-git --skip-tests

ng new AzureRouting --routing false --style css --skip-git --skip-tests

import {MatTabsModule} from '@angular/material/tabs';

ng new Chapter25 Routing and Navigation Part 1 --routing false --style css --skip-git --skip-tests

ng new Chapter19UsingServices --routing false --style css --skip-git --skip-tests

ng new etchitsketch --routing false --style css --skip-git --skip-tests

@NgModule({ declarations: [AppComponent], imports: [BrowserModule], providers: [], bootstrap: [AppComponent] })

**app.component.ts**

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

title = 'example';

}

The properties for the @Component decorator configure its behavior. The selector property tells Angular that this component will be used to replace an HTML element called app-root. The templateUrl and styleUrls properties tell Angular that the HTML content that the component wants to present to the user can be found in a file called app.component.html and that the CSS styles to apply to the HTML content are defined in a file called app.component.css (although the CSS file is empty in new projects).

The Mozilla Foundation provides a useful reference for all the objects that are used to represent HTML elements in the DOM at developer.mozilla.org/en-US/docs/Web/API . For each element, Mozilla provides a summary of the properties that are available and what each is used for. Start with HTMLElement ( developer.mozilla.org/en-US/docs/Web/API/HTMLElement ), which provides the functionality common to all elements.

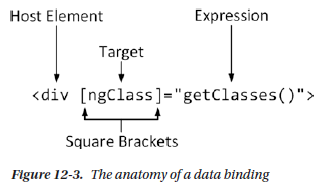
The built-in directives can be recognized by the ng prefix. The binding in Listing 12-2 gives you a hint that the target is a built-in directive that is related to the class membership of the host element.

***Listing 12-2.*** The Contents of the template.html File in the src/app Folder

<div [ngClass]="getClasses()" >

Hello, World.

</div>



<div [ngClass]="'text-white m-2 p-2 ' + getClasses()" >

A data binding has these four parts:

• The *host element* is the HTML element that the binding will affect, by changing its

appearance, content, or behavior.

• The *square brackets* tell Angular that this is a one-way data binding. When Angular

sees square brackets in a data binding, it will evaluate the expression and pass the

result to the binding’s *target* so that it can modify the host element.

• The *target* specifies what the binding will do. There are two different types of target: a

*directive* or a *property binding*.

• The *expression* is a fragment of JavaScript that is evaluated using the template’s

component to provide context, meaning that the component’s property and methods

can be included in the expression, like the getClasses method in the example

binding.

Table 12-3. The Basic Built-in Angular Directives

ngClass This directive is used to assign host elements to classes, as described in the “Setting Classes and Styles” section

<div [ngClass]="'text-white m-2 p-2 ' + getClasses()">

Name: {{ model.getProduct(1)?.name || 'None' }}

</div>

The string interpolation binding is denoted using pairs of curly brackets ( {{ and }} ). A single element can contain multiple string interpolation bindings.

The ngSwitch binding responds to changes in the data model, which you can test by executing the following statements in the browser’s JavaScript console: model.products.shift()

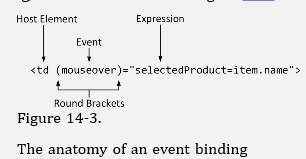
appRef.tick()

The ngFor directive assigns the variable to each object in the data source so that it is available for use by the nested elements. The local template variable in the example is called item , and it is used to access the Product object’s properties for the td elements, like this:

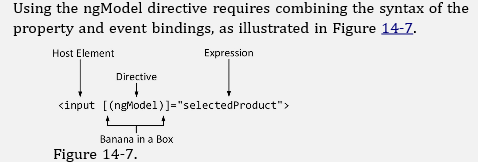
<tr \*ngFor="let item of getProducts() "> = <td>{{ item.name }}</td>

<tr \*ngFor="let item of getProducts(); let i = index "> = <td>{{ i + 1 }}</td>

Figure 14-3



<input class="form-control" [(ngModel)]="selectedProduct" />



The anatomy of a two-way data binding

A combination of square and round brackets is used to denote a two-way data binding, with the round brackets placed inside the square ones: [( and )] . The Angular development team refers to this as the banana-in-a-box binding because that’s what the brackets and parentheses look like when placed like this [()]

An Observer is created that collects the events and distributes them to subscribers

through an Observable. The simplest way to achieve this is to create a Subject,

which provides both Observer and Observable functionality. The flow of events to a

subscriber can be managed using a set of operators.

Page 579

The key Reactive Extensions building block is an Observable, which represents an observable sequence of

events. An object such as a component can subscribe to an Observable and receive a notification each time

an event occurs, allowing it to respond only when the event has been observed instead of every time there is

a change anywhere in the application.

Page 581

You may be used to representing asynchronous activities using Promise objects. Observables perform

the same basic role but are more flexible and have more features. Angular does provide support for

working with Promise objects, which can be useful when you transition to Angular and when working

with libraries that rely on Promise objects.

YourInitials220410

Your global Angular CLI version (10.0.0) is greater than your local

version (9.0.7). The local Angular CLI version is used.

To disable this warning use "ng config -g cli.warnings.versionMismatch false".

**Pages of important interests:**

Pages 327, 405