Angular
Binding, Routing, Directives

.NET

Data-binding is a mechanism used for coordinating what users see with what the Component contains.

Modeling – Data Binding

_____

The double curly braces ({{ }}) are *Angular's* interpolation binding syntax. Interpolation binding presents the component's property *values* inside the accompanying HTML Doc.

Property binding with [] around the property to be bound. This is one-way.

```
[class.selected]="hero === selectedHero"
```

Event binding based on events like 'click' or 'hover' to methods in the .ts file) using ().

```
<button (click)="addToCart(product)">Buy</button>
```

Two-Way Binding.

```
<input [(ngModel)]="hero.name" placeholder="name"/>
```

Two-Way Data Binding

[(ngModel)] is Angular's two-way *data binding* syntax. It *binds* the property to the HTML so that data can flow in both directions.
@ngModule *decorators* have the metadata needed for an Angular app to function. The most important @NgModule *decorator* annotates the top-level *AppModule* class.

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

To use forms, in app.module.ts import *FormsModule*, then add *FormsModule* to the imports array in the same file.

```
imports: [

    BrowserModule,

    FormsModule

],
```

Class Binding

You can add and remove CSS class designations from an element with *class binding*.

To create a single *class binding*, start with the prefix 'class' followed by '.nameOfCssClass' ([class.selected]="condition") .

Angular adds the class label when the bound expression is *truthy*, and it removes the class label when the expression is *falsy*.

```
[class.selected]="hero===selectedHero"
```

Angular Templates - Event Binding

The parentheses around click tell *Angular* to listen for the <li> element's click event. When the user clicks in the <li> element, *Angular* executes the onSelect(hero) expression on the element.

```
<li *ngFor="let hero of heroes" (click)="onSelect(hero)">
```

In this example, the **structural directive** *ngFor will create a <li> for each **hero** object in the **heroes** collection. Each <li> will have a click event attached to that **hero** and submit that **hero** as an argument to the onSelect() function.

Modeling – Decorators

Decorators are functions that allow a service, directive or filter to be modified prior to its usage. *Decorators* always begin with a @. They do not alter the original code.

Angular has *Class* and *Field* types of decorators:

Type	Decorator Name	Purpose
Class Decorators	@Component()	Marks a class as a component and provides configuration metadata.
	@Directive()	Attaches specific behavior to elements in the DOM
	@Pipe()	Supplies configuration metadata.
	@Injectible()	Marks a class as available for Dependency Injection.
	@NgModule()	Marks a class as a *Module* and supplies config metadata.
Field Decorators	@Input	Marks class fields as input properties and supplies config metadata. An input property is bound to a DOM property in the template and is updated with the DOM property's value.
	@Output	Marks class fields as output properties and supplies config metadata. The DOM property bound to the output property is auto-updated.

Component Decorator

@Component - This decorator indicates that the following class is a component. It provides the *selector, templateUrl*, and *styleUrls* metadata.

- The *selector* is a unique identifier for the component. It is the name used when the *component* is nested in a parent *component template*.
- The *templateUrl*, and *styleUrls* reference the relative HTML and CSS file locations generated for the component.

```
@Component({
  selector: 'app-player-list',
  templateUrl: './player-list.component.html',
  styleUrls: ['./player-list.component.css']
})
```

Structural Directives

Structural directives shape or reshape the DOM's structure, typically by adding, removing, and manipulating the elements to which they are attached. Directives with an asterisk, *, are *structural directives*.

```html
<div *ngIf="hero" class="name">{{hero.name}}</div>


<ul>
  <li *ngFor="let hero of heroes">{{hero.name}}</li>
</ul>


<div [ngSwitch]="hero?.emotion">
  <app-happy-hero    *ngSwitchCase="'happy'"   [hero]="hero"></app-happy-hero>
  <app-sad-hero      *ngSwitchCase="'sad'"     [hero]="hero"></app-sad-hero>
  <app-confused-hero *ngSwitchCase="'confused'" [hero]="hero"></app-confused-hero>
  <app-unknown-hero  *ngSwitchDefault          [hero]="hero"></app-unknown-hero>
</div>
```

Angular Routing

A *route* associates URL paths with a *component*.

Register a new *route* in app.module.ts or in an app-routing.module file.

The routerLink directive in the component .html template gives the *router* control over the *anchor* element. Insert routerLink into an element when you want to redirect to another (registered) URL.

```
const routes: Routes = [
  { path: 'heroes', component: HeroesComponent }
];
```

```
routerLink="/heroes/{{hero.id}}"
```

Angular Routing

Routes tell the *Router* which view to display when a user clicks a link.

A typical Angular *Route* has two properties:

- path: a string that matches the URL in the browser address bar.

- component: the component that the router should create when navigating to this route.

@NgModule metadata initializes the router and starts it listening for browser location changes.

```
@NgModule({
    imports: [RouterModule.forRoot(routes)],
    exports: [RouterModule]
})
```

The forRoot() method supplies the service providers and directives needed for routing and performs the initial navigation based on the current browser URL

Routing Step-by-step

1. Add a module called app-routing with
   - ng generate module app-routing --flat --module=app
2. Make sure *RouterModule* and *Routes* are imported into app-routing.module with
   - import { RouterModule, Routes } from '@angular/router';
   - Also import whatever *component* (from its relative location) you will be routing to into app-routing.module.ts
3. Delete CommonModule references and Declarations array.
4. Configure routes in const routes: Routes = [{ path:'link', component: AssociatedComponent }]; in *app-routing.module*
5. Add imports: [RouterModule.forRoot(routes)], under @NgModule.
6. Under @NgModule add exports: [RouterModule].
7. In app.component.html, where you want all route html templates to appear, add:
   -
8. Add <a routerLink="/[link]">NameOfLink to whatever page you want to add a link to.

```
1  import { HeroDetailComponent } from './hero-detail/hero-detai
2  import { NgModule } from '@angular/core';
3  import { RouterModule, Routes } from '@angular/router';
4  import { DashboardComponent } from './dashboard/dashboard.com
5  import { HeroesComponent } from './heroes/heroes.component';
6
7  const routes: Routes = [
8    { path: '', redirectTo: '/dashboard', pathMatch: 'full' },
9    { path: 'heroes', component: HeroesComponent },
10   { path: 'dashoard', component: DashboardComponent },
11   { path: 'detail/:id', component: HeroDetailComponent }
12
13 ];
14
15 @NgModule({
16   imports: [RouterModule.forRoot(routes)],
17   exports: [RouterModule]
18 })
19 export class AppRoutingModule { }
20
```