Class 12 - The Basics

Class 12 Course Content

View Class 12 Demo Online Here

Lesson Outline

Today we will learn:

1. How to Create Components from scratch
2. How to Create Components using the CLI
3. How to Style our Components
4. The Basics of Data-Binding
5. The Basics of Event-Binding
6. How to Use Two-Way-Binding
7. How to Use String Interpolation
8. The Basics of Structural Directives
9. The Basics of Attribute Directives

Lesson Notes

- **Component:** A *component* is a section or feature of your application. Every component has its own template, style, and logic. The benefit is that they are reusable and controllable.
- **Data-Binding:** *Data-Binding* is how we automatically update our pages template when our application state changes.
- **Event-Binding:** *Event-Binding* is a one-way connection from the view/template to the data source/logic.
- **Two-Way-Binding:** *Two-Way-Binding* is when we connect the view/template to the source/logic, and when either of them changes, they both update.

Components & Databinding Project Steps

- *Note*: This begins with a new project w/ bootstrap hooked up.

STEP 1: Creating a Component from Scratch

- *Note*: You should start class with a project created w/ bootstrap hook up.

box/box.component.ts file:

- Right-click on the "app" folder and create a new fold title "box".
- Inside the "box" folder, create a new file title "box.component.ts".

- Export a class titled "BoxComponent" (all words caps).
- Add the `Component({})` decorator. (Make sure to import this from "@angular/core").
- Add the configuration for the component by adding a "selector" property that points to a string value "app-box".
- Add another property in the "Component" decorator titled "templateUrl" that is a key to the value of "./box.component.html".

```
import { Component } from "@angular/core";

@Component({
  selector: "app-box",
  templateUrl: "./box.component.html",
})
export class BoxComponent {}
```

box/box.component.html file:

- Create the "box.component.html" file inside the "box" folder.
- Add a `<div class="card mt-3 box">` with a `<p class="text-center">I am a box!</p>` inside.

```
<div class="card mt-3 box">
  <p class="text-center">I am a box!</p>
</div>
```

app.module.ts file:

- *Note*: To use this component in our app, we must import it and add it to the declarations array in our main "app.modules.ts" file.

```
import { NgModule } from "@angular/core";
import { FormsModule } from "@angular/forms";
import { BrowserModule } from "@angular/platform-browser";

import { AppComponent } from "./app.component";
import { BoxComponent } from "./box/box.component";

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

app.component.html file:

- First, create a container, row, and column that takes up the full width. Put an `<h1>` inside.
- Since we chose the string "app-box" for our selector, we can use that tag inside of our HTML.

```
<div class="container mt-5">
  <div class="row justify-content-center">
    <div class="col-xs-12">
      <h1>Angular Boxes</h1>
      <app-box></app-boxes>
    </div>
  </div>
</div>
```

STEP 2: Creating a Component Using the CLI

Terminal:

- Create a new component using the CLI that will hold multiple "app-box" components. Title it "boxes"

```
ng g c boxes
```

- Delete the test file.

boxes/boxes.component.html file:

- Create three instances of the "app-box" component using the `<app-box></app-box>` tags.

```
<app-box></app-box>
<app-box></app-box>
<app-box></app-box>
```

app.component.html file:

- Change the tag to render the `<app-boxes>` instead of a singular `<app-box>`.

STEP 3: Styling the Components

app.component.css file:

- Style the `<h1>` tag to be "red", have a size of "4rem" and a margin-bottom of ".5em".

```
h1 {
  color: crimson;
  font-size: 4rem;
  margin-bottom: 0.5em;
}
```

box/box.component.css file:

- Create the "box.component.css" file inside the "box" folder.
- Style the ".box" class by adding a background-color, color, font-size, height, and flex properties.

```
.box {
  background-color: saddlebrown;
  color: white;
  font-size: 1.5rem;
  height: 200px;
  display: flex;
  align-items: center;
  justify-content: center;
}
```

box/box.component.ts file:

- Link your newly created styles by adding another property in the "Component({})" Decorator.

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

STEP 4: Outputting Dynamic Content (Databinding)

box/box.component.ts file:

- Create two new variables: `boxNumber = 2;` and `isEmpty = false;`

box/box.component.html file:

- Output these variables in the box markup inside two paragraph elements. (Walkthrough the ternary operator slowly)!

```
<p class="text-center">I am box number {{ boxNumber }}!</p>
<p>I am {{ isEmpty ? "empty" : "full" }}.</p>
```

boxes/boxes.component.ts file:

- Create a new boolean variable: `canOpenMoreBoxes = false`.
- Inside the "ngOnInit", create a timeout function that sets the "canOpenMoreBoxes" boolean variable to true after 3 seconds.

```
export class BoxesComponent implements OnInit {
  canOpenMoreBoxes = false;

  constructor() {}

  ngOnInit(): void {
    setTimeout(() => {
      this.canOpenMoreBoxes = true;
    }, 3000);
  }
}
```

boxes/boxes.component.html file:

- Create a button above the boxes that allows us to create a new box. (This button should be disabled if "canOpenMoreBoxes" is false).

```
<button class="btn btn-primary" [disabled]="!canOpenMoreBoxes">
  Open New Box
</button>
```

boxes/boxes.component.ts file:

- Create another variable: `boxMockText = "You should open a box man."`.
- Create a function that changes the "boxMockText" whenever you open a new box. Name this function accordingly.

```
export class BoxesComponent implements OnInit {
  canOpenMoreBoxes = false;
  boxMockText = "You haven't opened a box in a while";

  constructor() {}

  ngOnInit(): void {
    setTimeout(() => {
      this.canOpenMoreBoxes = true;
    }, 3000);
  }
```

```
  onOpenBox() {
    this.boxMockText = "You just opened another box!";
  }
}
```

boxes/boxes.component.html file:

- Create a paragraph tag below the button that outputs the "boxMockText" variable string.
- Add a `(click)` event listener on the button that calls our "onOpenBox()" function.
- Remove the "boxNumber" paragraph.

```
<button
  class="btn btn-primary"
  [disabled]="!canOpenMoreBoxes"
  (click)="onOpenBox()"
>
  Open New Box
</button>
<p>{{ boxMockText }}</p>
<hr />
<!-- . . . -->
```

boxes/boxes.component.ts file:

- Create a new variable: `boxName = "Default Box";`
- Inside the "onOpenBox()" function, change the string to a template literal (``) and add in the "boxName" variable.

```
boxName = "Default Box"

// . . .

onOpenBox() {
  this.boxMockText = `You just opened a box called: ${this.boxName}`
}
```

boxes/boxes.component.html file:

- Create a label and input that binds to the "boxName" variable.
- *Note*: Make sure you have the { FormsModule } imported in your application.

```
<label for="boxName">Box Name:</label>
<input type="text" class="form-control mb-3" [(ngModel)]="boxName" />
<button
```

```
  class="btn btn-primary"
  [disabled]="!canOpenMoreBoxes"
  (click)="onOpenBox()"
>
  Open New Box
</button>
<!-- . . . -->
```

STEP 5: Using Angular Directives

boxes/boxes.component.html file:

- Add an `*ngIf` directive to the `{{ boxMockText }}` paragraph and bind it to a boolean varaible "haveOpenedABox".

```
<p *ngIf="haveOpenedABox">{{boxMockText}}</p>
```

boxes/boxes.component.ts file:

- Create the "haveOpenedABox" variable and set it to initialize as false.
- In the "onOpenBox()" function, set the "haveOpenedABox" to true.

```
haveOpenedABox = false;
// . . .
onOpenBox() {
  // . . .
  haveOpenedABox = true;
}
```

box/box.component.ts file:

- Create a constructor. Inside, set our "isEmpty" variable to be true half the time and false the other half using math.
- Create a function that gets a color depending on our "isEmpty" variable.

```
constructor() {
  this.isEmpty = Math.random() > 0.5;
}

getColor() {
  return this.isEmpty === true ? 'red' : 'green';
}
```

box/box.component.html file:

- Using the "isEmpty" variable, display different text with a different class depending on the value.

```
<p>
  I am
  <span [ngStyle]="{ color: getColor() }">{{ isEmpty ? "empty" : "full" }}
</span
  >.
</p>
```

- We can do this in a different way using the `[ngClass]` Directive. We will show this by placing a "ngClass" attribute on the card div.

```
<div class="card" [ngClass]="{ emptyBox: isEmpty === true }"></div>
```

box/box.component.css file:

- Create style for the ".emptyBox" class.

```
.emptyBox {
  height: 100px;
  font-size: 1rem;
  opacity: 0.75;
}
```

boxes/boxes.component.ts file:

- Create an array called "boxes" and fill it with dummy data.
- Inside the "onOpenBox()" function, push the current boxName to the "boxes" array.

```
export class BoxesComponent implements OnInit {
  canOpenMoreBoxes = false;
  hasOpenedABox = false;
  boxMockText = "";
  boxName = "";
  boxes = ["Box 1", "Box 2", "Box 3"];

  // . . .

  onOpenBox() {
    this.hasOpenedABox = true;
    this.boxes.push(this.boxName);
    this.boxMockText = `You just opened a box called: "${this.boxName}"!`;
```

```
  }
}
```

boxes/boxes.component.html file:

- Remove all but on `<app-box>` tag.
- Place an "*ngFor" loop on the `<app-box>` that loops over all the boxes in the "boxes" array.

```
<app-box *ngFor="let box of boxes"></app-box>
```

Additional Notes

Class Exercise

1. Generate a new Angular project (with strict mode disabled). Name it "angular-basics-exercise".
2. Manually create a component called "article".
3. In the article component, create variables `title: string = "Whatever you want"` and `content: string = "Some content goes here"`.
4. Use string interpolation to output the title in an h1 element and the content in a paragraph element. (Note: you can google 'lorem ipsum generator ' to generate dummy text).
5. Display the article component by adding it in the main "app.component.html" file.
6. In the article component, add another variable: `isTechRelated: boolean = true`.
7. Use "ngStyle" to change the background of the h1 element to blue when "isTechrelated" is true; Otherwise, it should be yellow.
8. Push your local repository to GitHub, call the repo 'angular-basics-exercise'.

Bonus: In the article component, create a button with the content "change isTechRelated". (Use event binding to reverse the variable "isTechRelated" value whenever a user clicks the button).

Bonus: In the article component, use "ngIf" to display a paragraph element with content "Tech Related" when the "isTechRelated" is true. When false, use the ng-template element to display "Not Tech Related".

Resources

- Angular Docs - Binding Syntax
- Angular Docs - Built-in Directives
- Angular Docs - Attribute Directives

- Angular Docs - Structural Directives