



Codergirl – Frontend

Unit 2 - Class 3

December 7, 2020

# Agenda

- Components and Databinding Deep dive
- Studio – Assignment 4
- Holiday break is 12/28 to 12/30, with the last class being 12/23 and the first of new year being on 1/4/21.

# Splitting the app into Components

npm i

ng serve

ng g c cockpit

ng g c server-element

# Cockpit component

Copy the first row into cockpit.component.html

```
<div class="row">
  <div class="col-xs-12">
    <p>Add new Servers or blueprints!</p>
    <label>Server Name</label>
    <input type="text" class="form-control" [(ngModel)]="newServerName">
    <label>Server Content</label>
    <input type="text" class="form-control" [(ngModel)]="newServerContent">
    <br>
    <button
      class="btn btn-primary"
      (click)="onAddServer()">Add Server</button>
    <button
      class="btn btn-primary"
      (click)="onAddBlueprint()">Add Server Blueprint</button>
  </div>
</div>
```

related methods in ts  
as well as

newServerName = ";

newServerContent = ";

```
c > app > cockpit > TS cockpit.component.ts >  CockpitComponent >  newSe
1   import { Component, OnInit } from '@angular/core';
2
3   @Component({
4     selector: 'app-cockpit',
5     templateUrl: './cockpit.component.html',
6     styleUrls: ['./cockpit.component.css']
7   })
8   export class CockpitComponent implements OnInit {
9
10   constructor() { }
11
12   ngOnInit(): void {
13   }
14   newServerName = '';
15   newServerContent = '';
16
17   onAddServer() {
18     this.serverElements.push({
19       type: 'server',
20       name: this.newServerName,
21       content: this.newServerContent
22     });
23   }
24
25   onAddBlueprint() {
26     this.serverElements.push({
27       type: 'blueprint',
28       name: this.newServerName,
29       content: this.newServerContent
30     });
31   }
32
33 }
```

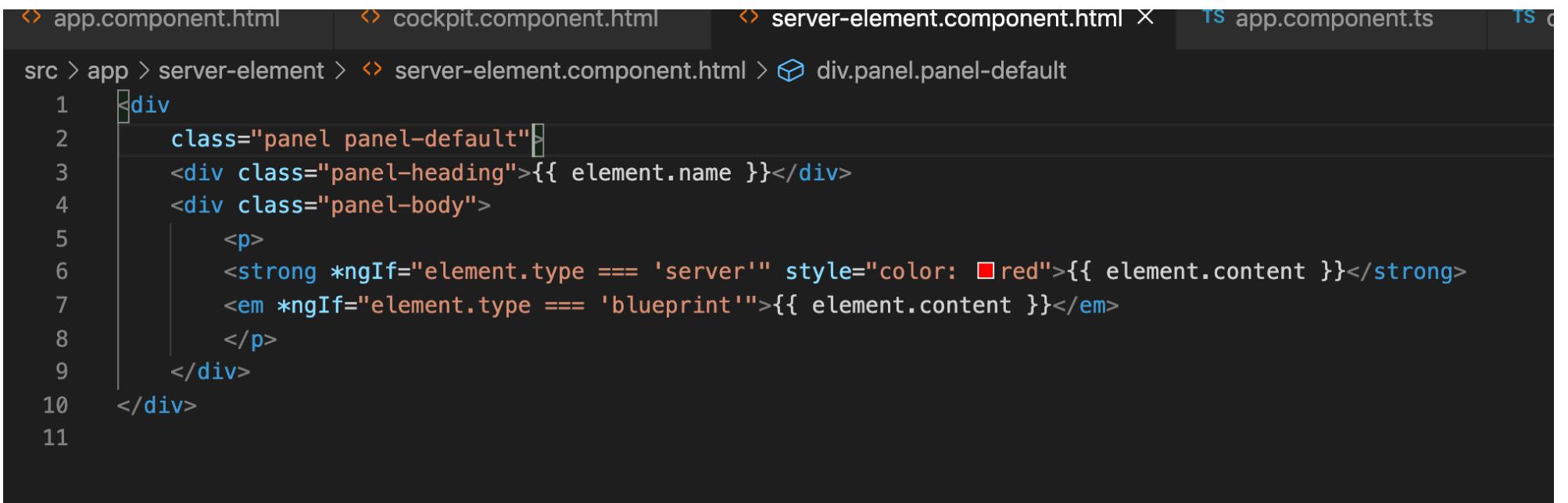
# Server-element Component

Move the row into server-element.html

```
> app > server-element > server-element.component.html > ...
<div
  class="panel panel-default"
  *ngFor="let element of serverElements">
  <div class="panel-heading">{{ element.name }}</div>
  <div class="panel-body">
    <p>
      <strong *ngIf="element.type === 'server'" style="color: red">{{ element.content }}</strong>
      <em *ngIf="element.type === 'blueprint'">{{ element.content }}</em>
    </p>
  </div>
</div>
```

# Server-element Component

Since server-element is a single server we remove the ngFor



The screenshot shows a code editor with multiple tabs at the top: app.component.html, cockpit.component.html, server-element.component.html (selected), app.component.ts, and another partially visible tab. The server-element.component.html tab contains the following code:

```
src > app > server-element > server-element.component.html > div.panel.panel-default
1 <div
2   class="panel panel-default"
3   <div class="panel-heading">{{ element.name }}</div>
4   <div class="panel-body">
5     <p>
6       <strong *ngIf="element.type === 'server'" style="color: red">{{ element.content }}</strong>
7       <em *ngIf="element.type === 'blueprint'">{{ element.content }}</em>
8     </p>
9   </div>
10 </div>
11
```

The code defines a single server element using a standard `<div>` tag instead of an `<ngFor>` loop.

# app-component.html

Add app-cockpit and app-server-element and  
ngFor for serverElements

```
src/app/app-component.html:1...  
1  <div class="container">  
2    <app-cockpit></app-cockpit>  
3    <hr>  
4    <div class="row">  
5      <div class="col-xs-12">  
6        <app-server-element *ngFor="let serverElement of serverElements"></app-server-element>  
7      </div>  
8    </div>  
9  </div>  
10 |
```

# App is broke!

The screenshot shows a terminal window with the following interface elements:

- Top bar: TERMINAL, PROBLEMS (2), OUTPUT, DEBUG CONSOLE.
- Right side: A dropdown menu labeled "1: node" with icons for closing, opening, and saving.

The terminal output is as follows:

```
Date: 2020-12-05T01:58:31.560Z - Hash: a24b7dcbabdb5a5df3cb
6 unchanged chunks

Time: 56ms
: Compiled successfully.

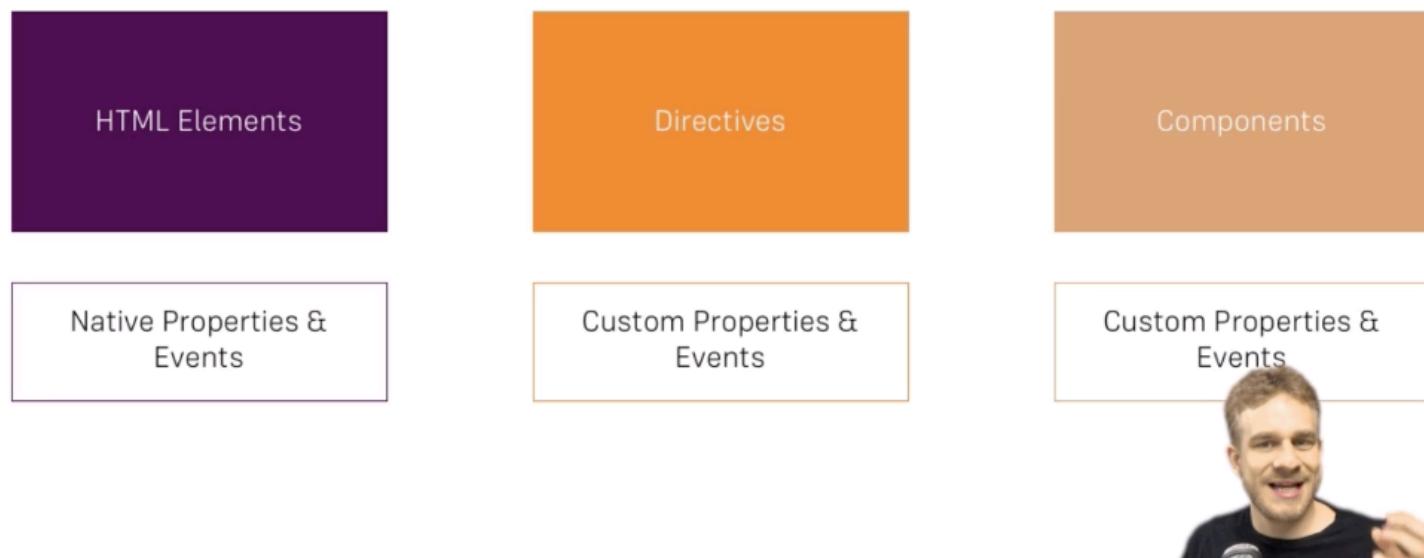
ERROR in src/app/cockpit/cockpit.component.ts:18:10 - error TS2339: Property 'serverElements' does not exist on type 'CockpitComponent'.
18     this.serverElements.push({
          ~~~~~
src/app/cockpit/cockpit.component.ts:26:10 - error TS2339: Property 'serverElements' does not exist on type 'CockpitComponent'.
26     this.serverElements.push({
          ~~~~~
```

# Need to pass data between Components

Property and Event binding to pass data into  
Elements

They can also be used on directives and  
Components.

# Need to pass data between Components



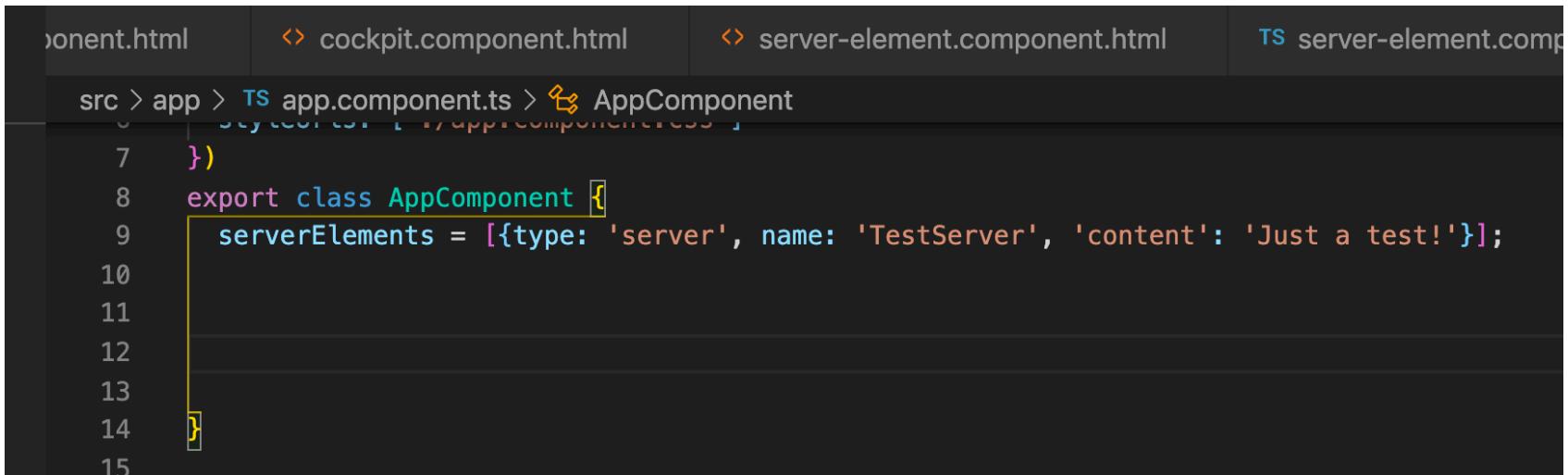
# Comment out code that isn't compiling

```
15
16
17     onAddServer() {
18         // this.serverElements.push({
19         //   type: 'server',
20         //   name: this.newServerName,
21         //   content: this.newServerContent
22         // });
23     }
24
25     onAddBlueprint() {
26         // this.serverElements.push({
27         //   type: 'blueprint',
28         //   name: this.newServerName,
29         //   content: this.newServerContent
30         // });
31     }
32
33 }
34
```

# server-element.html has element

```
./app/server-element/server-element.component.ts ...
1 import { Component, OnInit } from '@angular/core';
2
3 @Component({
4   selector: 'app-server-element',
5   templateUrl: './server-element.component.html',
6   styleUrls: ['./server-element.component.css']
7 })
8 export class ServerElementComponent implements OnInit {
9
10   element: {type: string, name: string, content: string}
11   constructor() { }
12
13   ngOnInit(): void {
14   }
15
16 }
17
```

# app-component.ts



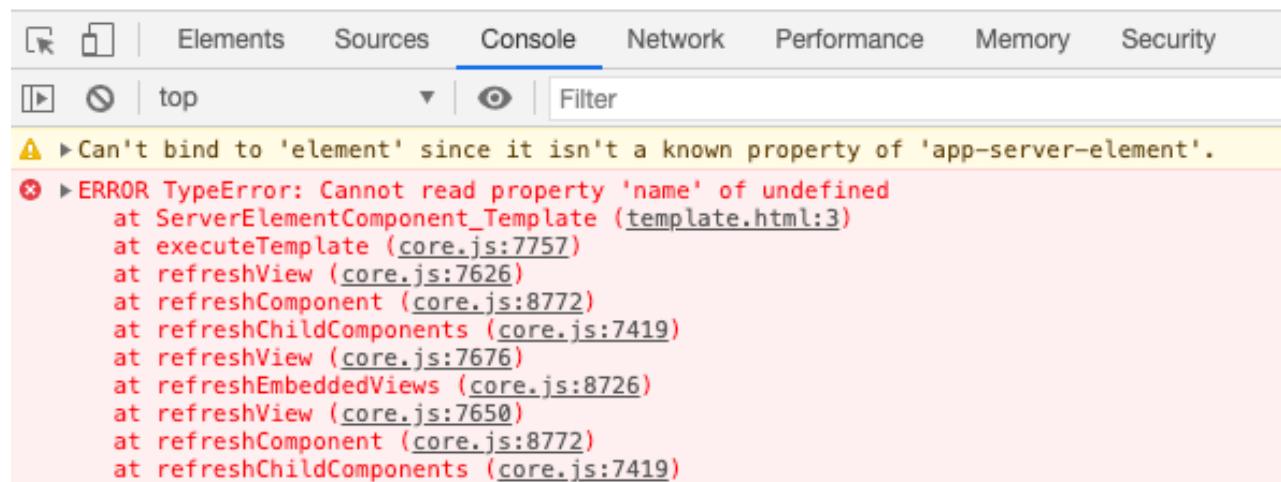
```
src > app > TS app.component.ts > AppComponent
  ...
  7  })
  8  export class AppComponent {
  9    serverElements = [{type: 'server', name: 'TestServer', content: 'Just a test!'}];
 10
 11
 12
 13
 14 }
 15
```

A screenshot of a code editor showing the file 'app-component.ts'. The code defines a class 'AppComponent' with a single property 'serverElements' set to an array containing one object. A yellow rectangular selection highlights the entire class definition from line 8 to line 14. The code editor interface includes tabs for 'cockpit.component.html', 'server-element.component.html', and 'server-element.component.css'.

Properties are not exposed outside of the component

# app-component.ts

```
<div class="container">
  <app-cockpit></app-cockpit>
  <hr>
  <div class="row">
    <div class="col-xs-12">
      <app-server-element
        *ngFor="let serverElement of serverElements"
        [element]="serverElement"></app-server-element>
    </div>
  </div>
</div>
```



# app-component.ts

Need to add a decorator for the property to be accessible outside

```
src > app > server-element > TS server-element.component.ts > ...
1  import { Component, OnInit, Input } from '@angular/core';
2
3  @Component({
4    selector: 'app-server-element',
5    templateUrl: './server-element.component.html',
6    styleUrls: ['./server-element.component.css']
7 })
8  export class ServerElementComponent implements OnInit {
9
10    @Input() element: {type: string, name: string, content: string}
11    constructor() { }
12
13    ngOnInit(): void {
14    }
15
16  }
17
```

# Now app loads!

A screenshot of a web browser window displaying a server configuration interface. The URL bar shows `http://localhost:4200`. The page header includes a back arrow, forward arrow, and refresh icon. Below the header is a navigation bar with links: Apps, real-world-vue, LaunchCode, Dev tools, Bayer dev tools, Buying, Fitness, INSPIRE, Vue learning, Bayer Sites, LEPSI, Bayer, and Personal.

The main content area has a heading "Add new Servers or blueprints!" followed by two input fields: "Server Name" and "Server Content". Below these fields are two buttons: "Add Server" and "Add Server Blueprint".

At the bottom, there is a list of servers, with "TestServer" currently selected. The "TestServer" card displays the message "Just a test!" in red text.

# ...also supports alias

```
<div class="container">
  <app-cockpit></app-cockpit>
  <hr>
  <div class="row">
    <div class="col-xs-12">
      <app-server-element
        *ngFor="let serverElement of serverElements"
        [srvElement]="serverElement"></app-server-element>
    </div>
  </div>
</div>
```

```
import { Component, OnInit, Input } from '@angular/core';

@Component({
  selector: 'app-server-element',
  templateUrl: './server-element.component.html',
  styleUrls: ['./server-element.component.css']
})
export class ServerElementComponent implements OnInit {

  @Input('srvElement') element: {type: string, name: string, content: string}
  constructor() { }

  ngOnInit(): void {}
}
```

# Informing parent component

```
TS app.component.ts ×  
src > app > TS app.component.ts > 🏃 AppComponent  
1 import { Component } from '@angular/core';  
2  
3 @Component({  
4   selector: 'app-root',  
5   templateUrl: './app.component.html',  
6   styleUrls: ['./app.component.css']  
7 })  
8 export class AppComponent {  
9   serverElements = [{type: 'server', name: 'TestServer', 'content': 'Just a test!'}];  
10  
11   onServerAdded(serverData: {serverName: string, serverContent: string}) {  
12     this.serverElements.push(  
13       type: 'server',  
14       name: serverData.serverName,  
15       content: serverData.serverContent  
16     );  
17   }  
18  
19   onBlueprintAdded(blueprintData: {serverName: string, serverContent: string}) {  
20     this.serverElements.push(  
21       type: 'blueprint',  
22       name: blueprintData.serverName,  
23       content: blueprintData.serverContent  
24     );  
25   }  
26 }  
27
```

# Informing parent component

```
TS cockpit.component.ts ×

src > app > cockpit > TS cockpit.component.ts > ...
1 import { Component, OnInit, EventEmitter, Output } from '@angular/core';
2
3 @Component({
4   selector: 'app-cockpit',
5   templateUrl: './cockpit.component.html',
6   styleUrls: ['./cockpit.component.css']
7 })
8 export class CockpitComponent implements OnInit {
9   serverCreated = new EventEmitter<{serverName: string, serverContent: string}>();
10  @Output() blueprintCreated = new EventEmitter<{serverName: string, serverContent: string}>();
11  @Output() newServerName = '';
12  newServerContent = '';
13
14  constructor() { }
15
16  ngOnInit(): void {
17  }
18
19  onAddServer() {
20    |   (property) serverName: string
21    |   this.serverCreated.emit({serverName: this.newServerName, serverContent: this.newServerContent});
22  }
23
24  onAddBlueprint() {
25    |   this.blueprintCreated.emit({serverName: this.newServerName, serverContent: this.newServerContent});
26  }
27}
```

# App works!

← → ⌂ ⓘ http://localhost:4200

\_apps real-world-vue LaunchCode Dev tools Bayer dev tools Buying Fitness INSPIRE Vue learning Bayer Sites LEPSI Bayer Personal

Add new Servers or blueprints!

**Server Name**  
server2

**Server Content**  
server2 rocks

**Add Server** **Add Server Blueprint**

TestServer

**Just a test!**

server1

**server1 rocks**

server2

**server2 rocks**

# Alias works for output too.

```
TS cockpit.component.ts ×

src > app > cockpit > TS cockpit.component.ts > CockpitComponent
1   import { Component, OnInit, EventEmitter, Output } from '@angular/core';
2
3   @Component({
4     selector: 'app-cockpit',
5     templateUrl: './cockpit.component.html',
6     styleUrls: ['./cockpit.component.css']
7   })
8   export class CockpitComponent implements OnInit {
9     @Output() serverCreated = new EventEmitter<{serverName: string, serverContent: string}>();
10    @Output('bpCreated') blueprintCreated = new EventEmitter<{serverName: string, serverContent: string}>();
11    newServerName = '';
12    newServerContent = '';
13
14    constructor() { }
15
16    ngOnInit(): void {
17    }
18
19    onAddServer() {
20      this.serverCreated.emit({serverName: this.newServerName, serverContent: this.newServerContent});
21    }
22
23    onAddBlueprint() {
24      this.blueprintCreated.emit({serverName: this.newServerName, serverContent: this.newServerContent});
25    }
26
27  }
28
```

# Alias works for output too.

```
TS cockpit.component.ts      TS app.component.ts ×  
src > app > TS app.component.ts > 🏛 AppComponent > 📁 onBlueprintAdded  
4   selector: 'app-root',  
5   templateUrl: './app.component.html',  
6   styleUrls: ['./app.component.css']  
7 }  
8 export class AppComponent {  
9   serverElements = [{type: 'server', name: 'TestServer', 'content': 'Just a test!'}];  
10  
11  onServerAdded(serverData: {serverName: string, serverContent: string}) {  
12    this.serverElements.push({  
13      type: 'server',  
14      name: serverData.serverName,  
15      content: serverData.serverContent  
16    });  
17  }  
18  
19  onBlueprintAdded(blueprintData: {serverName: string, serverContent: string}) {  
20    this.serverElements.push({  
21      type: 'blueprint',  
22      name: blueprintData.serverName,  
23      content: blueprintData.serverContent  
24    });  
25  }  
26 }  
27
```

# App works

← → ⌂ ⓘ http://localhost:4200 ★

Apps real-world-vue LaunchCode Dev tools Bayer dev tools Buying Fitness INSPIRE Vue learning Bayer Sites LEPSI Bayer Personal

Add new Servers or blueprints!

**Server Name**  
test blueprint

**Server Content**  
test blueprint content

**Add Server** **Add Server Blueprint**

---

TestServer

Just a test!

---

test blueprint

*test blueprint content*

# View Encapsulation

Styles in CSS only get applied to Components they belong to.

Angular enforces style encapsulation.

All elements have a unique attribute names.

Similar to Shadow DOM – each element has it's own. Not supported by all browsers. Hence Angular implements it.

Encapsulation can be overridden. None is used to have global CSS. Default is Emulated and Native isn't supported by all browsersss.

# View Encapsulation

```
TS server-element.component.ts ×

src > app > server-element > TS server-element.component.ts > ...
1 import { Component, OnInit, Input, ViewEncapsulation } from '@angular/core';
2
3 @Component({
4   selector: 'app-server-element',
5   templateUrl: './server-element.component.html',
6   styleUrls: ['./server-element.component.css'],
7   encapsulation: ViewEncapsulation.None
8 })
9 export class ServerElementComponent implements OnInit {
10
11   @Input('srvElement') element: {type: string, name: string, content: string}
12   constructor() { }
13
14   ngOnInit(): void {
15   }
16
17 }
18
```

# View Encapsulation

```
src > app > server-element > # server-element.component.css > label
1   p {
2     |   color: blue;
3   }
4
5   label {
6     |   color: red;
7 }
```

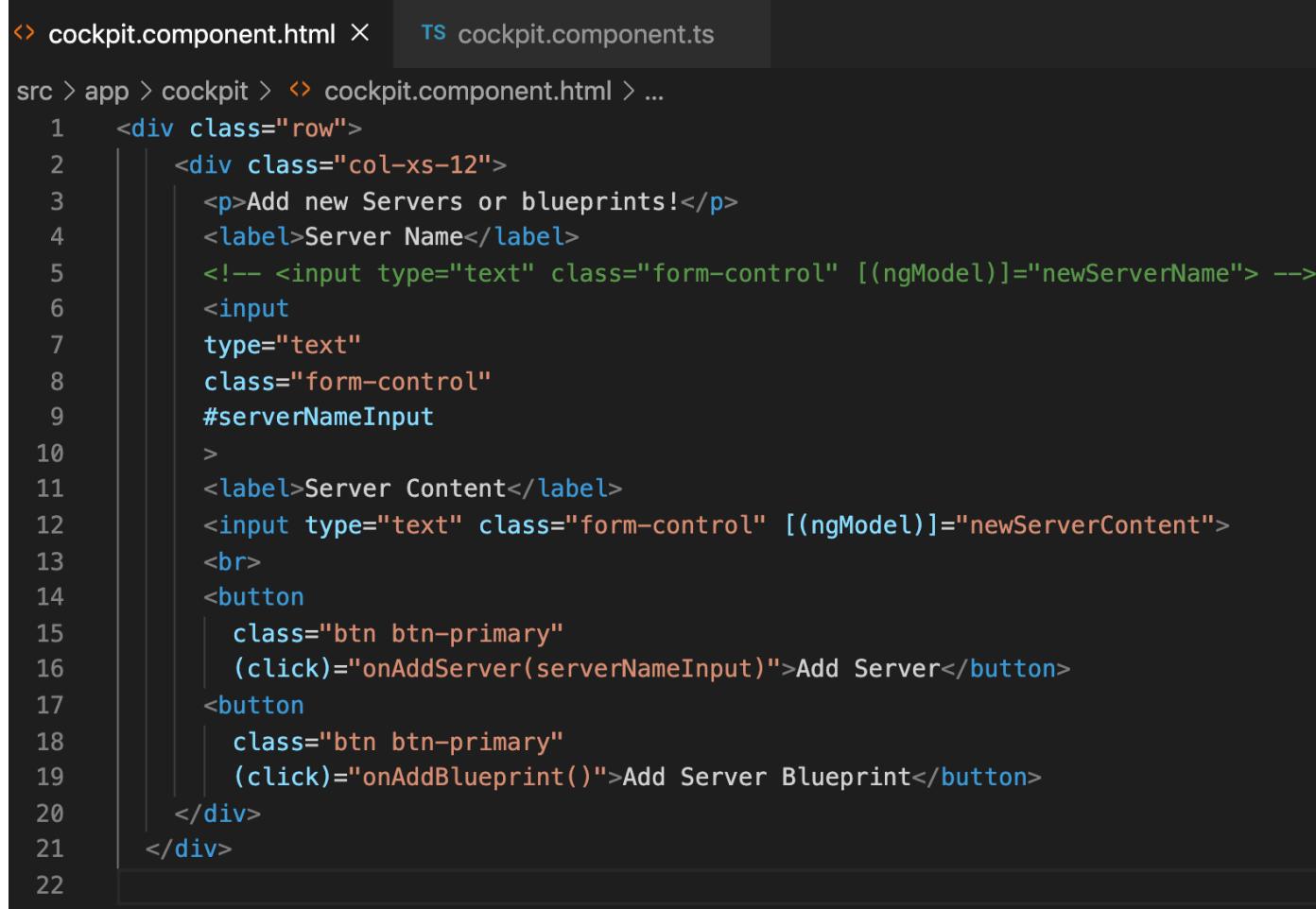
# View Encapsulation

A screenshot of a web browser window displaying a server configuration interface. The URL bar shows `http://localhost:4200`. The page has a header with various links: Apps, real-world-vue, LaunchCode, Dev tools, Bayer dev tools, Buying, Fitness, INSPIRE, Vue learning, Bayer Sites, LEPSI, Bayer, and Personal. Below the header, there's a section titled "Add new Servers or blueprints!" with fields for "Server Name" and "Server Content". At the bottom of this section are two buttons: "Add Server" (highlighted in blue) and "Add Server Blueprint". Below these buttons is a list of servers, with "TestServer" currently selected. The "TestServer" card displays the text "Just a test!". The overall layout uses a clean, modern design with a light gray background and white cards for each server entry.

# Local Reference

Instead of two-way data binding, we can place a local reference on an element.

It can be applied to any element.



The screenshot shows a code editor with two tabs: 'cockpit.component.html' and 'cockpit.component.ts'. The 'cockpit.component.html' tab is active, displaying the following code:

```
src > app > cockpit > cockpit.component.html > ...
1  <div class="row">
2    <div class="col-xs-12">
3      <p>Add new Servers or blueprints!</p>
4      <label>Server Name</label>
5      <!-- <input type="text" class="form-control" [(ngModel)]="newServerName"> -->
6      <input
7        type="text"
8        class="form-control"
9        #serverNameInput
10       >
11      <label>Server Content</label>
12      <input type="text" class="form-control" [(ngModel)]="newServerContent">
13      <br>
14      <button
15        class="btn btn-primary"
16        (click)="onAddServer(serverNameInput)">Add Server</button>
17      <button
18        class="btn btn-primary"
19        (click)="onAddBlueprint()">Add Server Blueprint</button>
20    </div>
21  </div>
```

# Local Reference

Instead of two-way data binding, we can place a local reference on an element.

It can be applied to any element.

```
src > app > cockpit > TS cockpit.component.ts > CockpitComponent > constructor
1   import { Component, OnInit, EventEmitter, Output } from '@angular/core';
2
3   @Component({
4     selector: 'app-cockpit',
5     templateUrl: './cockpit.component.html',
6     styleUrls: ['./cockpit.component.css']
7   })
8   export class CockpitComponent implements OnInit {
9     @Output() serverCreated = new EventEmitter<{serverName: string, serverContent: string}>();
10    @Output('bpCreated') blueprintCreated = new EventEmitter<{serverName: string, serverContent: string}>();
11    newServerName = '';
12    newServerContent = '';
13
14    constructor() { }
15
16    ngOnInit(): void {
17    }
18
19    onAddServer(nameInput) {
20      console.log(nameInput.value);
21      this.serverCreated.emit({serverName: this.newServerName, serverContent: this.newServerContent});
22    }
23
24    onAddBlueprint() {
25      this.blueprintCreated.emit({serverName: this.newServerName, serverContent: this.newServerContent});
26    }
27
28  }
29
```

# Local Reference

The screenshot shows a web browser window with the URL `http://localhost:4200` in the address bar. The page content is a form for adding a new server or blueprint. It has fields for 'Server Name' (containing 'jyuy') and 'Server Content' (containing 'fhh'). Below these fields are two buttons: 'Add Server' (highlighted in blue) and 'Add Server Blueprint'. To the right of this form, there is a preview section titled 'TestServer' containing the text 'Just a test!'. At the bottom of the page, the Angular development mode message is visible: 'Angular is running in the development mode. Call enableProdMode() to enable the production mode.' and '[WDS] Live Reloading enabled.' The browser's developer tools are open at the bottom, specifically the 'Console' tab, which shows the command 'jyuy' entered.

# Local Reference

```
src > app > cockpit > TS cockpit.component.ts > CockpitComponent
1 import { Component, OnInit, EventEmitter, Output } from '@angular/core';
2
3 @Component({
4   selector: 'app-cockpit',
5   templateUrl: './cockpit.component.html',
6   styleUrls: ['./cockpit.component.css']
7 })
8 export class CockpitComponent implements OnInit {
9   @Output() serverCreated = new EventEmitter<{serverName: string, serverContent: string}>();
10  @Output('bpCreated') blueprintCreated = new EventEmitter<{serverName: string, serverContent: string}>();
11  newServerName = '';
12  newServerContent = '';
13
14  constructor() { }
15
16  ngOnInit(): void {
17  }
18
19  onAddServer(nameInput: HTMLInputElement) {
20    console.log(nameInput.value);
21    this.serverCreated.emit({serverName: nameInput.value, serverContent: this.newServerContent});
22  }
23
24  onAddBlueprint(nameInput: HTMLInputElement) {
25    this.blueprintCreated.emit({serverName: nameInput.value, serverContent: this.newServerContent});
26  }
}
```

# Local Reference

← → ⌂ ⓘ http://localhost:4200

Apps real-world-vue LaunchCode Dev tools Bayer dev tools Buying Fitness INSPIRE Vue learning Bayer Sites LEPSI Bayer Personal

Add new Servers or blueprints!

**Server Name**

CVV

**Server Content**

vvb

Add Server Add Server Blueprint

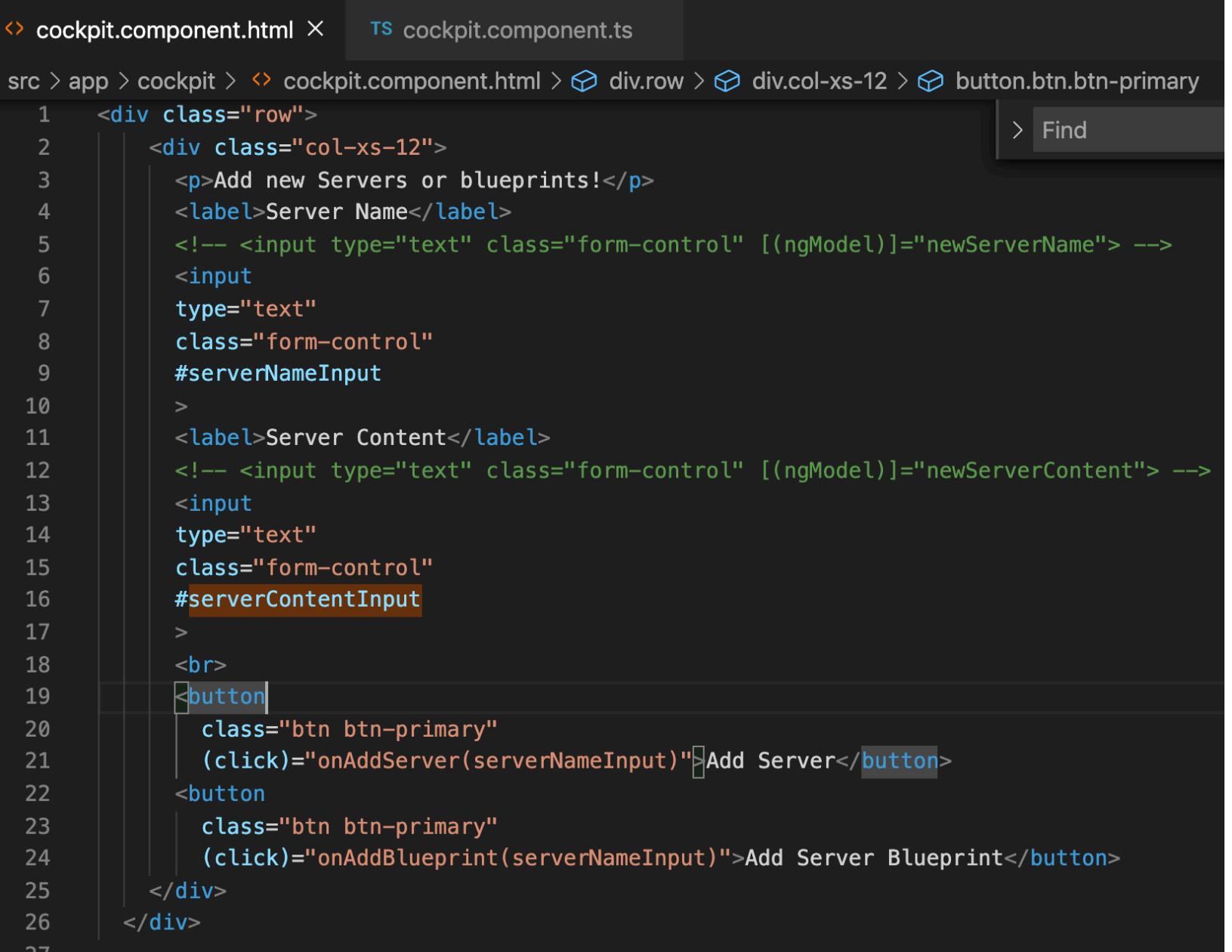
TestServer

Just a test!

CVV

vvb

# ViewChild

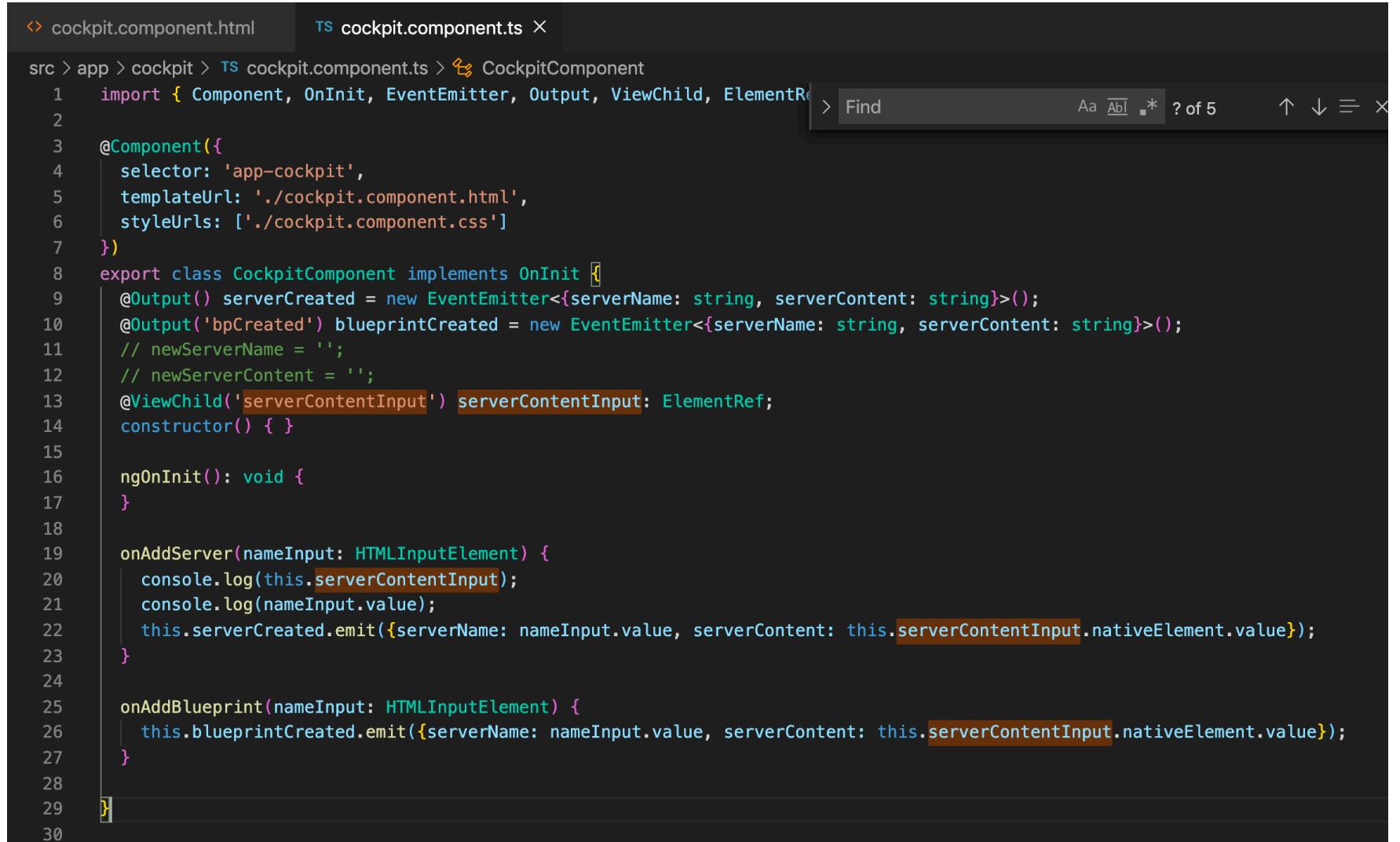


The screenshot shows a code editor with two tabs: 'cockpit.component.html' and 'cockpit.component.ts'. The 'cockpit.component.html' tab is active, displaying the following HTML code:

```
<div class="row">
  <div class="col-xs-12">
    <p>Add new Servers or blueprints!</p>
    <label>Server Name</label>
    <!-- <input type="text" class="form-control" [(ngModel)]="newServerName"> -->
    <input
      type="text"
      class="form-control"
      #serverNameInput
    >
    <label>Server Content</label>
    <!-- <input type="text" class="form-control" [(ngModel)]="newServerContent"> -->
    <input
      type="text"
      class="form-control"
      #serverContentInput
    >
    <br>
    <button
      class="btn btn-primary"
      (click)="onAddServer(serverNameInput)">Add Server</button>
    <button
      class="btn btn-primary"
      (click)="onAddBlueprint(serverNameInput)">Add Server Blueprint</button>
  </div>
</div>
```

The code editor interface includes a status bar at the top with file paths and a search bar. A floating 'Find' dialog box is visible on the right side of the editor area.

# ViewChild



The screenshot shows a code editor interface with two tabs: 'cockpit.component.html' and 'cockpit.component.ts'. The 'cockpit.component.ts' tab is active, displaying the following TypeScript code:

```
src > app > cockpit > ts cockpit.component.ts > CockpitComponent
1 import { Component, OnInit, EventEmitter, Output, ViewChild, ElementRef } from '@angular/core';
2
3 @Component({
4   selector: 'app-cockpit',
5   templateUrl: './cockpit.component.html',
6   styleUrls: ['./cockpit.component.css']
7 })
8 export class CockpitComponent implements OnInit {
9   @Output() serverCreated = new EventEmitter<{serverName: string, serverContent: string}>();
10  @Output('bpCreated') blueprintCreated = new EventEmitter<{serverName: string, serverContent: string}>();
11  // newServerName = '';
12  // newServerContent = '';
13  @ViewChild('serverContentInput') serverContentInput: ElementRef;
14  constructor() { }
15
16  ngOnInit(): void {
17  }
18
19  onAddServer(nameInput: HTMLInputElement) {
20    console.log(this.serverContentInput);
21    console.log(nameInput.value);
22    this.serverCreated.emit({serverName: nameInput.value, serverContent: this.serverContentInput.nativeElement.value});
23  }
24
25  onAddBlueprint(nameInput: HTMLInputElement) {
26    this.blueprintCreated.emit({serverName: nameInput.value, serverContent: this.serverContentInput.nativeElement.value});
27  }
28
29 }
30
```

The code demonstrates the use of the `@ViewChild` decorator to inject an `ElementRef` for a specific input element named `'serverContentInput'`. This reference is then used to log the element and its value to the console, and to emit an event with the element's value.

# ViewChild

The screenshot shows a web application interface. At the top, there is a header bar with a back arrow, forward arrow, refresh icon, and a link to <http://localhost:4200>. Below the header is a navigation bar with links to various Bayer-related sites like Apps, real-world-vue, LaunchCode, Dev tools, and others.

The main content area contains a form for adding a server. It has two input fields: "Server Name" with value "jhgj" and "Server Content" with value "kukl". Below the inputs are two buttons: "Add Server" and "Add Server Blueprint".

Below the form, there are three separate sections, each representing a server entry:

- TestServer**: Contains the text "Just a test!"
- jhgj**: Contains the text "kukl"
- jhgj**: Contains the text "kukl" (in blue)

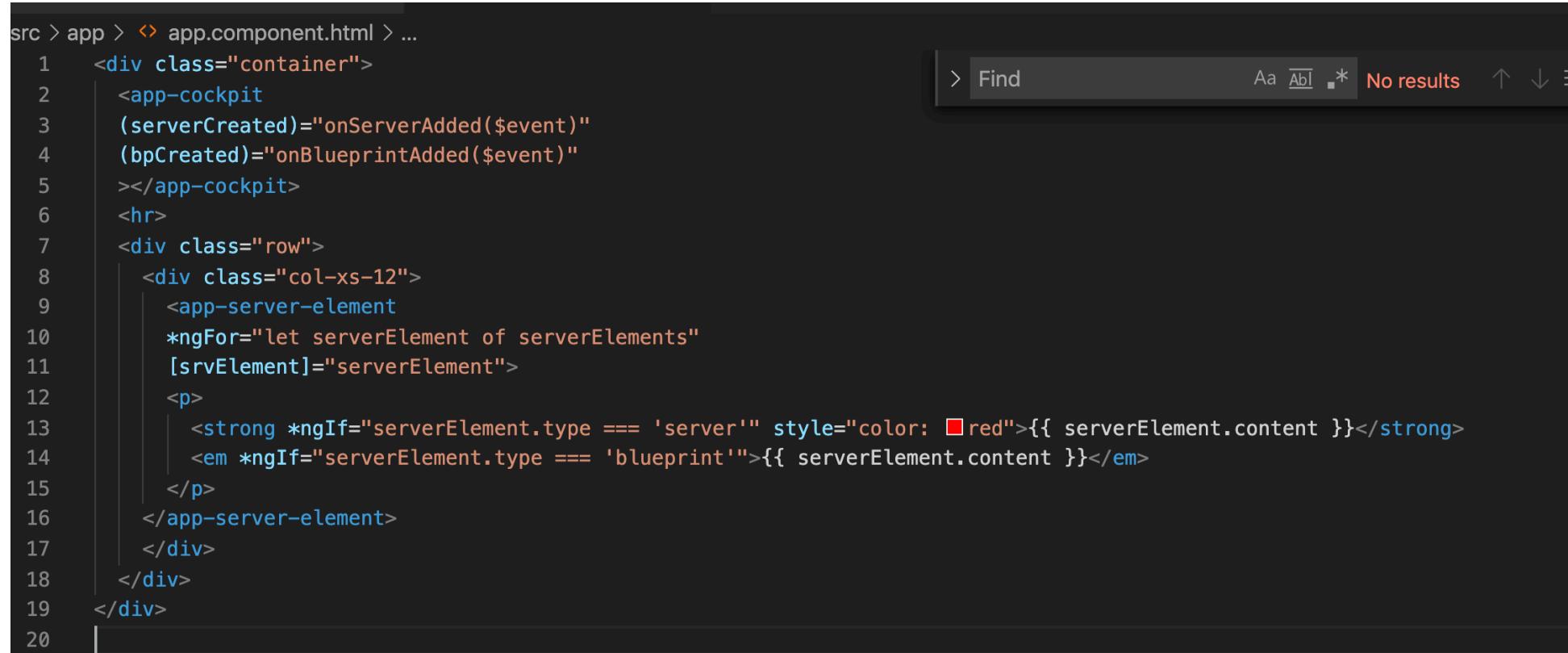
At the bottom of the page is a DevTools console tab. The tabs include Elements, Sources, **Console**, Network, Performance, Memory, Security, Application, Lighthouse, and Augury. The Console tab is active, showing the following output:

```
Angular is running in the development mode. Call enableProdMode() to enable the production mode.  
[WDS] Live Reloading enabled.  
▶ ElementRef {nativeElement: input.form-control}  
jhgj  
> |
```

Don't access the DOM and modify using ViewChild!

# ng-content

Add the content into a Component using ng-content, that it finds between opening and closing tag.



The screenshot shows a code editor interface with a dark theme. The file being edited is `app.component.html`. The code uses the `ng-content` selector to insert dynamic content into the component's template. The code is as follows:

```
src > app > app.component.html > ...
1  <div class="container">
2    <app-cockpit
3      (serverCreated)="onServerAdded($event)"
4      (bpCreated)="onBlueprintAdded($event)"
5    ></app-cockpit>
6    <hr>
7    <div class="row">
8      <div class="col-xs-12">
9        <app-server-element
10       *ngFor="let serverElement of serverElements"
11       [srvElement]="serverElement">
12        <p>
13          <strong *ngIf="serverElement.type === 'server'" style="color: red">{{ serverElement.content }}</strong>
14          <em *ngIf="serverElement.type === 'blueprint'">{{ serverElement.content }}</em>
15        </p>
16        </app-server-element>
17      </div>
18    </div>
19  </div>
20 |
```

The code editor has a search bar at the top right with the placeholder "Find". Below the search bar are buttons for font size (Aa), font style (Ab), and a refresh icon. To the right of those are the messages "No results" and navigation icons for up, down, and left.

# ng-content

```
<> server-element.component.html × <> app.component.html  
src > app > server-element > <> server-element.component.html > ...  
1   <div  
2     class="panel panel-default">  
3       <div class="panel-heading">{{ element.name }}</div>  
4       <div class="panel-body">  
5         <ng-content></ng-content>  
6       </div>  
7   </div>  
8
```

# ng-content

← → ⌂ ⓘ http://localhost:4200

\_apps real-world-vue LaunchCode Dev tools Bayer dev tools Buying Fitness INSPIRE Vue learning Bayer Sites LEPSI Bayer Personal

Add new Servers or blueprints!

**Server Name**  
vb

**Server Content**  
nmnn

**Add Server** **Add Server Blueprint**

TestServer

Just a test!

vb

nmnn

# Component Lifecycle

## Lifecycle

ngOnChanges	Called after a bound input property changes
ngOnInit	Called once the component is initialized
ngDoCheck	Called during every change detection run
ngAfterContentInit	Called after content (ng-content) has been projected into view
ngAfterContentChecked	Called every time the projected content has been checked
ngAfterViewInit	Called after the component's view (and child views) has been initialized
ngAfterViewChecked	Called every time the view (and child views) have been checked
ngOnDestroy	Called once the component is about to be destroyed



# Component Lifecycle

```
's server-element.component.ts ×  
src > app > server-element > TS server-element.component.ts > ServerElementComponent > con  
2   Component,  
3   OnInit,  
4   Input,  
5   ViewEncapsulation,  
6   OnChanges,  
7   SimpleChanges  
8 } from '@angular/core';  
9  
10 @Component({  
11   selector: 'app-server-element',  
12   templateUrl: './server-element.component.html',  
13   styleUrls: ['./server-element.component.css'],  
14   encapsulation: ViewEncapsulation.None  
15 })  
16 export class ServerElementComponent implements OnInit, OnChanges {  
17  
18   @Input('srvElement') element: {type: string, name: string, content: string}  
19   constructor() {  
20     console.log('constructor called');  
21   }  
22  
23   ngOnChanges(changes: SimpleChanges) {  
24     console.log('ngOnChanges called');  
25     console.log(changes);  
26   }  
27  
28   ngOnInit(): void {  
29     console.log('ngOnInit called');  
30   }  
31 }  
32 }
```

# Component Lifecycle

The screenshot shows a web browser at `http://localhost:4200`. The page has a header with links to various services like Apps, real-world-vue, LaunchCode, Dev tools, Bayer dev tools, Buying, Fitness, and INSP. The main content area has a heading "Add new Servers or blueprints!" and two input fields: "Server Name" and "Server Content". Below these is a button bar with "Add Server" and "Add Server Blueprint". Underneath, there's a section labeled "TestServer" containing the text "Just a test!". The entire interface is styled with a light gray background and red text for labels.

The screenshot shows the Chrome DevTools Console tab. The tabs at the top are Elements, Sources, Console, Network, Performance, Memory, Security, Application, and Lighthouse. The Console tab is selected. The log output shows the following messages:

```
constructor called
ngOnChanges called
  ▼ {element: SimpleChange}
    ▼ element: SimpleChange
      ► currentValue: {type: "server", name: "TestServer", content: "Just a test!"}
        firstChange: true
        previousValue: undefined
      ► __proto__: Object
      ► __proto__: Object
ngOnInit called
Angular is running in the development mode. Call enableProdMode() to enable the production mode.
[WDS] Live Reloading enabled.
```

# Component Lifecycle

Review code for server-component.ts

# Assignment

npm i

ng serve

ng g c gameControl

ng g c odd

ng g c even

# Questions?

# Studio!