

Workshop. Introduction to Angular Elements

Contents

Contents.....1

Explanation of Colors.....2

Task 01. Adding Angular Elements .....3

Task 02. Adding A New Component .....4

Task 03. Creating a Custom Element .....5

Task 04. Building for Production.....6

Task 05. Creating a Build Script.....7

Task 06. Using an Element .....8

Task 07. Creating a Button w/ Style as Element .....9

Task 08. Creating a Login Form as Element .....10

Task 09. Using an Element Input .....12

Task 10. Using an Element Output .....13

Task 11. Using ElementRef, @HostBinding, @HostListener.....14

Task 12. Using a Content Projection .....15

Task 13. Using Slots.....16

## Explanation of Colors

Blue color is a snippet of code that you need to fully use to create a new file.

Black color in combination with green or red, means the snippet of code that was added earlier.

Green color is the snippet of code that needs to be added.

Red color is the snippet of code that needs to be removed.

## Task 01. Adding Angular Elements

1. Run the following command from the command line to add Angular Elements functionality to the project

```
>> ng add @angular/elements --project=angularElements*
```

\*May be it is not always necessary to use parameter --project

## Task 02. Adding A New Component

1. Create **SimpleComponent**. Run the following command from the command line to add Angular Elements functionality to the project

>> **ng g c simple -v ShadowDom**

2. Add Bootstrap CSS to the component. Modify the file **simple.component.css**. Use the following snippet of CSS:

```
@import '~bootstrap/dist/css/bootstrap.min.css';
```

3. Make changes to the **AppModule**. Use the following snippet of code:

```
// 1
import { AppComponent } from './app.component';

// 2
declarations: [
  AppComponent,
  SimpleComponent
],
bootstrap: [AppComponent]
```

4. Remove **AppComponent** from the project.
5. Make changes to the **index.html**. Use the following snippet of HTML:

```
<app-root></app-root>
<app-simple></app-simple>
```

## Task 03. Creating a Custom Element

1. Make changes to **AppModule**. Use the following snippet of code:

```
// 1
import { NgModule, Injector, DoBootstrap } from '@angular/core';
import { createCustomElement } from '@angular/elements';

// 2
entryComponents: [SimpleComponent]

// 3
export class AppModule implements DoBootstrap {
  constructor(private injector: Injector) {}

// 4
  ngDoBootstrap() {
    // All this code can be placed to the constructor function, leaving this function empty.
    const el = createCustomElement(SimpleComponent, { injector: this.injector });
    // Name of custom element must include dash!
    customElements.define('app-simple', el);

    // This component exists in the global register of custom elements
    // We get function.
    console.log(customElements.get('app-simple'));
    // This component doesn't exist in the global register of custom elements.
    // We get undefined.
    console.log(customElements.get('app-root'));
  }
}
```

## Task 04. Building for Production

1. Run the following command from the command line to build your project

```
>> ng build --prod --output-hashing none
```

## Task 05. Creating a Build Script

1. In the top-level project folder create a new file with name **elements-build.js** and insert the following code:

```
const fs = require('fs-extra');
const concat = require('concat');

(async function build() {
  const files = [
    './dist/angularElements/runtime.js',
    './dist/angularElements/polyfills.js',
    './dist/angularElements/scripts.js',
    './dist/angularElements/main.js',
  ]

  await fs.ensureDir('elements');

  await concat(files, 'elements/simple.js');

  await fs.copyFile('./dist/angularElements/styles.css', 'elements/styles.css');

  await fs.copy('./dist/angularElements/assets/', 'elements/assets/');
})();
```

2. Add the following line to the script section of **package.json**:

```
"scripts": {
  ...,
  "build:elements": "ng build --prod --output-hashing none && node elements-build.js"
},
```

3. Run the following command from the command line:

```
>> npm run build:elements
```

## Task 06. Using an Element

1. Create a **elements/index.html** file. Use the following snippet of HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Angular Elements </title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <app-simple></app-simple>
  <script src="simple.js"></script>
</body>
</html>
```

2. Open this file in a browser.

### Notice!

If there is an issue to run index.html, install document-register-element v1.8.1



## Task 07. Creating a Button w/ Style as Element

1. Replace the **SimpleComponent Template**. Use the following snippet of HTML:

```
<button class="btn btn-primary">  
  button works!  
</button>
```

2. Run the following command from the command line:

```
>> npm run build:elements
```

3. Create a **elements/index.html** file. Use the following snippet of HTML:

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">  
  <meta name="viewport" content="width=device-width, initial-scale=1.0">  
  <title>Angular Elements </title>  
  <link rel="stylesheet" href="styles.css">  
</head>  
<body>  
  <app-simple></app-simple>  
  <script src="simple.js"></script>  
</body>  
</html>
```

4. Open the **elements/index.html** in a browser.

## Task 08. Creating a Login Form as Element

1. Modify the **AppModule**. Use the following snippet of code:

```
// 1
import { ReactiveFormsModule } from '@angular/forms';

// 2
imports: [BrowserModule, ReactiveFormsModule],
```

2. Replace the **SimpleComponent Class**. Use the following snippet of code:

```
import { Component, OnInit, ViewEncapsulation } from '@angular/core';
import { FormGroup, FormBuilder, Validators } from '@angular/forms';

@Component({
  selector: 'app-simple',
  templateUrl: './simple.component.html',
  styleUrls: ['./simple.component.css'],
  encapsulation: ViewEncapsulation.ShadowDom
})
export class SimpleComponent implements OnInit {
  loginForm: FormGroup;

  constructor(private fb: FormBuilder) {}

  ngOnInit() {
    this.buildLoginForm();
  }

  onLogin() {
    console.log(this.loginForm.value);
  }

  private buildLoginForm() {
    this.loginForm = this.fb.group({
      email: ['', Validators.required],
      password: ['', Validators.required],
      checkMe: false
    });
  }
}
```

3. Replace the **SimpleComponent Template**. Use the following snippet of code:

```
<button class="btn btn-primary">
  button works!
</button>

<form (ngSubmit)="onLogin()" [formGroup]="loginForm">
  <div class="form-group">
    <label for="email">Email address</label>
    <input type="email" class="form-control" id="email"
      formControlName="email"
      aria-describedby="emailHelp" placeholder="Enter email">
    <small id="emailHelp" class="form-text text-muted">We'll never share your email with anyone
else.</small>
  </div>
  <div class="form-group">
```

```

    <label for="password">Password</label>
    <input type="password" class="form-control" id="password"
      formControlName="password"
      placeholder="Password">
  </div>
  <div class="form-group form-check">
    <input type="checkbox" class="form-check-input" id="checkMe"
      formControlName="checkMe">
    <label class="form-check-label" for="checkMe">Check me out</label>
  </div>
  <button type="submit"
    [disabled]=loginForm.invalid
    class="btn btn-primary">Submit</button>
</form>

```

4. Run the following command from the command line:

```
>> npm run build:elements
```

5. Create a **elements/index.html** file. Use the following snippet of HTML:

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Angular Elements </title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <app-simple></app-simple>
  <script src="simple.js"></script>
</body>
</html>

```

6. Open the **elements/index.html** in a browser. Fill the inputs and click submit button.

## Task 09. Using an Element Input

1. Make changes to the **SimpleComponent**. Use the following snippet of code:

```
// 1
import { Component, OnInit, ViewEncapsulation, Input } from '@angular/core';

// 2
@Input('email') emailDefaultValue = '';

// 3
private buildLoginForm() {
  this.loginForm = this.fb.group({
    email: [this.emailDefaultValue, Validators.required],
    password: ['', Validators.required],
    checkMe: false
  });
}
```

2. Run the following command from the command line:

>> **npm run build:elements**

3. Create a **elements/index.html** file. Use the following snippet of HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Angular Elements </title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <app-simple email="v.zhiritskiy@gmail.com"></app-simple>
  <script src="simple.js"></script>
</body>
</html>
```

4. Open the **elements/index.html** in a browser. The first field should have default value.

## Task 10. Using an Element Output

1. Make changes to the **SimpleComponent**. Use the following snippet of code:

```
// 1
import { Component, OnInit, ViewEncapsulation, Input, Output, EventEmitter } from '@angular/core';

// 2
@Output() sendLoginFormValue: EventEmitter<any> = new EventEmitter<any>();

// 3
onLogin() {
  console.log(this.loginForm.value);
  this.sendLoginFormValue.emit(this.loginForm.value);
}
```

2. Run the following command from the command line:

>> **npm run build:elements**

3. Create a **elements/index.html** file. Use the following snippet of HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Angular Elements </title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <app-simple id="simple" email="v.zhiritskiy@gmail.com"></app-simple>
  <script src="simple.js"></script>
  <script>
    const elem = document.getElementById('simple');
    elem.addEventListener('sendLoginFormValue', function(event) {
      console.log(event.detail);
    });
  </script>
</body>
</html>
```

4. Open the **elements/index.html** in a browser. Fill the form and click button Submit. Look to the console.

## Task 11. Using ElementRef, @HostBinding, @HostListener

1. Make changes to the **SimpleComponent**. Use the following snippet of code:

```
// 1
import { Component, OnInit, ViewEncapsulation, Input, Output, EventEmitter, ElementRef,
HostBinding, HostListener } from '@angular/core';

// 2
@HostBinding('attr.selected') isSelected = true;

// 3
constructor(private fb: FormBuilder, private el: ElementRef) {
  console.log(this.el.nativeElement);
}

// 4
@HostListener('click', ['$event'])
onHostClick(event) {
  console.log(
    'Click on Host Element. IsSelected (attr.selected): ',
    this.isSelected
  );
  console.log('Click on Host Element. $event: ', event);
}
```

2. Run the following command from the command line:

>> **npm run build:elements**

3. Create a **elements/index.html** file. Use the following snippet of HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Angular Elements </title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <app-simple id="simple" email="v.zhiritskiy@gmail.com"></app-simple>
  <script src="simple.js"></script>
  <script>
    const elem = document.getElementById('simple');
    elem.addEventListener('sendLoginFormValue', function(event) {
      console.log(event.detail);
    });
  </script>
</body>
</html>
```

4. Open the **elements/index.html** in a browser. Open the console. Click on the element.

## Task 12. Using a Content Projection

1. Make changes to the **SimpleComponent Template**. Use the following snippet of HTML:

```
<ng-content></ng-content>
<form (ngSubmit)="onLogin()" [formGroup]="loginForm">
```

2. Run the following command from the command line:

```
>> npm run build:elements
```

3. Create a **elements/index.html** file. Use the following snippet of HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Angular Elements </title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <app-simple id="simple" email="v.zhiritskiy@gmail.com">
    <h1>Login Form</h1>
  </app-simple>
  <script src="simple.js"></script>
  <script>
    const elem = document.getElementById('simple');
    elem.addEventListener('sendLoginFormValue', function(event) {
      console.log(event.detail);
    });
  </script>
</body>
</html>
```

4. Open the **elements/index.html** in a browser. You should see title “Login Form”.

## Task 13. Using Slots

1. Make changes to the **SimpleComponent Template**. Use the following snippet of HTML:

```
<ng-content></ng-content>

<form (ngSubmit)="onLogin()" [formGroup]="loginForm">
  <div class="form-group">
    <label for="email">Email address</label>
    <input type="email" class="form-control" id="email"
      formControlName="email"
      aria-describedby="emailHelp" placeholder="Enter email">
    <small id="emailHelp" class="form-text text-muted">We'll never share your email with anyone
else.</small>
  </div>
  <div class="form-group">
    <label for="password">Password</label>
    <input type="password" class="form-control" id="password"
      formControlName="password"
      placeholder="Password">
  </div>
  <div class="form-group form-check">
    <input type="checkbox" class="form-check-input" id="checkMe"
      formControlName="checkMe">
    <label class="form-check-label" for="checkMe">Check me out</label>
  </div>
  <button type="submit"
    [disabled]=loginForm.invalid
    class="btn btn-primary">Submit</button>
</form>
<button #el class="accordion" (click)="toggleHelper()">
  <slot name="header">Default header</slot>
</button>
<div class="panel">
  <slot name="details">Default details</slot>
</div>
```

2. Make changes to the **SimpleComponent Class**. Use the following snippet of code:

```
// 1
import {Component, OnInit, ViewEncapsulation, Input, Output, EventEmitter, ElementRef,
HostBinding, HostListener } from '@angular/core';
import { FormGroup, FormBuilder, Validators } from '@angular/forms';

// 2
export class SimpleComponent implements OnInit {
  // tslint:disable-next-line:no-input-rename
  @Input('email') emailDefaultValue = '';

  @Output() sendLoginFormValue: EventEmitter<any> = new EventEmitter<any>();

  @HostBinding('attr.selected') isSelected = true;

  loginForm: FormGroup;

  constructor(private fb: FormBuilder, private el: ElementRef) {
    console.log(this.el.nativeElement);
  }

  ngOnInit() {
```



```

    this.buildLoginForm();
  }

  @HostListener('click', ['$event'])
  onHostClick(event) {
    console.log(
      'Click on Host Element. IsSelected (attr.selected): ',
      this.isSelected
    );
    console.log('Click on Host Element. $event: ', event);
  }

  onLogin() {
    this.sendLoginFormValue.emit(this.loginForm.value);
  }

  private buildLoginForm() {
    this.loginForm = this.fb.group({
      email: [this.emailDefaultValue, Validators.required],
      password: ['', Validators.required],
      checkMe: false
    });
  }
}

export class SimpleComponent {
  @Input() icon = 'arrow';

  @ViewChild('el', { read: ElementRef }) el: ElementRef;

  toggleHelper() {
    this.el.nativeElement.classList.toggle('active');
    const panel = this.el.nativeElement.nextElementSibling;
    if (panel.style.maxHeight) {
      panel.style.maxHeight = null;
    } else {
      panel.style.maxHeight = `${panel.scrollHeight}px`;
    }
  }
}

```

3. Make changes to the SimpleComponent CSS. Use the following snippet of css:

```

@import '~bootstrap/dist/css/bootstrap.min.css';

.accordion {
  background-color: #eee;
  color: #444;
  cursor: pointer;
  padding: 18px;
  width: 100%;
  border: none;
  text-align: left;
  outline: none;
  font-size: 15px;
  transition: 0.4s;
}

.active, .accordion:hover {
  background-color: #ccc;
}

```

```

.accordion:after {
  content: '\002B';
  color: #777;
  font-weight: bold;
  float: right;
  margin-left: 5px;
}

.active:after {
  content: "\2212";
}

.panel {
  padding: 0 18px;
  background-color: white;
  max-height: 0;
  overflow: hidden;
  transition: max-height 0.2s ease-out;
}

```

4. Make changes to the **AppModule**. Use the following snippet of code:

```

// 1
constructor(private injector: Injector) {
  const el = createCustomElement(SimpleComponent, { injector });
  customElements.define('app-accordion', el);
}

// 2
ngDoBootstrap() {
  const el = createCustomElement(SimpleComponent, {
    injector: this.injector
  });
  customElements.define('app-simple', el);

  // This component exists in the global register of custom elements
  // We get function.
  console.log(customElements.get('app-simple'));
  // This component doesn't exist in the global register of custom elements.
  // We get undefined.
  console.log(customElements.get('app-root'));
}

```


5. Create a **elements/index.html** file. Use the following snippet of HTML:

```

<!DOCTYPE html>
<html>
  <head>
    <title>My WebPage</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">

  </head>
  <body>
    <app-accordion icon="plus">
      <span slot="header">A serious header</span>
      <span slot="details">With some crucial details</span>
    </app-accordion>
    <app-accordion icon="plus">
      <span slot="header">One more header</span>

```

```
    <span slot="details">I  web</span>
  </app-accordion>
  <script src="simple.js"></script>
</body>
</html>
```

6. Open the **elements/index.html** in a browser. You should see an accordion.