


Exercise 5 -Adding a second component

Objective

To build a new component that leverages Angular's data-binding and built in directives

Overview

In this exercise you are going to create an instructor gallery. Initially you will create the thumbnail gallery populated from a static array, and then you will react to click events to display more details of the individual 'clicked on' instructor. The end result should look something like this:


 transforming performance through learning


COURSES INSTRUCTORS LOCATIONS CONTACT US

Courses available at QA

Building JavaScript Applications Using NodeJS and the MEAN Stack QANODEDEV Classroom 3 days	Developing MVC single-page web applications using AngularJS QAANGULARJS Classroom 2 days	Developing Web Applications Using HTML5 QAIWEBUI Classroom 5 days
Leveraging the Power of JQuery QAJQUERY Classroom 3 days	Programming with JavaScript QAJAVASC Classroom 5 days	Next Generation JavaScript: ECMAScript2015 QAES2015 Classroom 2 days
Building Web Applications Using Angular QAANGULAR Classroom 3 days		

Instructors



**David Walker**

David is a change driven technologist who continually looks to adapt and expand his knowledge and understanding of his field. Over the last eighteen years David has led technology and training companies through emerging fields and technology trends helping them to understand the future and develop business opportunities. As Head of Emerging Technologies he works closely with customers and industry experts to ensure the opportunities and threats of new technology trends designing custom learning solutions to help small and enterprise organisation adapt and make the most of their people - ensuring QA is ready when our customers need to navigate the minefield of the fast moving digital landscape. His passion is in advanced web engineering principals and vendor neutral thick client design/development technologies reflected in his research, analysis and courseware development experience combined with his training delivery skills. As a technologist he is the lead instructor and syllabus author for web development technologies and specialising in Agile, DevOps, and User Experience driven approaches to developing solutions. He has authored courses such as HTML5, Responsive Web Development, User Experience, NodeJS, Javascript and JQuery.

© 1996-2017 QA Limited

Instructions

Part 1 - Creating a new feature module

1. Before we begin working on this new feature, we should create a new module. You can choose to do this either manually or using the Angular CLI.
 - a. Create a new module called instructors. This will encapsulate any and all code related to our instructors. If you are using the CLI then navigate to the root directory of the application for this exercise and use the following command

```
ng g module instructors
```

- b. If you are not using the CLI then please ensure you create this module in its own subfolder within the app folder

2. We now need to create the first component for this feature. Move to the instructors folder.
3. The Angular CLI can generate components on the fly for us, so you can either create a new component manually or, ensuring your CWD is the new feature's directory, run the following command

```
ng g component instructors-gallery
```

This will generate some boiler plate files for this component and import it into the new feature module.

4. Confirm the above has taken place by inspecting the files it generated and the new module's `imports` array; you should see the additions.
5. Check the component definition file to find the selector and then use this to place the component within the `template` of the root module. It should go at the bottom of the main element.
6. The root module does not know anything about this new feature -
 - a. Export the gallery component from the instructors module
 - b. Import the instructors module into the root module
7. Check that this all worked by running the application and verifying that the text "instructors-gallery works!" is just above the footer.

Part 2 - Creating the thumbnail gallery

8. Within the instructors folder create a new file called *instructor.model.ts* and export an Instructor class with the following properties defined:

firstname	string
surname	string
courses	string[]
location	string
bio	string
src	string

9. Import this class into the instructors gallery component
10. define a property within this component class called `selectedInstructor` of type `Instructor`.

11. Initialise a property called `instructors` of type `Instructor[]` and use the array found in `instructors.json` file within the `assets` folder to populate it.
12. We now need to create images for each instructor in this array.
 - a. Within the template file for our instructor gallery add a div with a class of "thumbnails"
 - b. Inside the div create an image element. Use an `ngFor` directive to create an `img` element for each instructor in the array, using the `src` property to construct the relevant url (you'll find the relevant images in the `assets` folder)
 - c. Inside the instructor gallery component `css` file add selectors for the gallery div and the images within. The images should each be given a `border-radius` of 50%. The gallery div should be given a `text-align` property with a value of center.

Part 3 - Adding the instructor detail view

13. We're now going to create an asset to display the bio of any instructor the user clicks on in our thumbnail gallery.
 - a. Back in the template file add a div below the current gallery div, give it a class of 'selected-instructor'
 - b. Within this div create an `h3`, `img` and `paragraph` element. This is where the details will go when the user clicks on an instructor's thumbnail picture.
14. The selected instructor's data for these elements will be stored in the `selectedInstructor` property of the component when a user clicks on the respective image. Set up data binding to display the appropriate heading, image and bio within the new elements you created.
15. When you save and view this in the browser you will get errors saying that `selectedInstructor` is undefined. This is expected given a user has not yet clicked on an instructor (besides, we haven't written the code yet!) so add an `ngIf` directive to the containing "selected-instructor" div to guard against this condition.
16. Add a click handler to the repeated image in the gallery. This click handler should call a function `selectInstructor`, passing the clicked instructor as the first argument.
17. Now write the function `selectInstructor()` in the component class so that it assigns the instructor to the `selectedInstructor` property.
18. Add the following CSS to the component's stylesheet to improve its aesthetics.

```
.selected-instructor {  
  display: flex;  
  flex-flow: row wrap;  
  align-items: center;  
}
```

```
.selected-instructor img {  
  flex: 0 0 auto;  
  border-radius: 50%;  
}  
  
.selected-instructor p {  
  flex: 1 1 0;  
  margin-left: 1em;  
  text-align: justify;  
}  
  
.selected-instructor h3 {  
  flex: 0 0 100%;  
}
```

19. Finally - use a style property binding to set the border of any of the clicked thumbnail images to `3px solid #003C70`
20. Test your page in the browser, it should look something like the image from the Overview.

Part 4 - Filtering the visible courses

In this part of the exercise you are going to use Output property bindings to alert the parent component to a change in selected instructor and the parent component will then filter the courses on display to only include ones that instructor teaches.

21. Start by adding an output property to the instructor gallery component called `instructorChange` and give it an alias of `change` and instantiate it as an `EventEmitter` that will emit `Instructor` types.
22. When a new instructor is selected we wish to emit that instructor from this `EventEmitter`. We already have a `selectInstructor` method in place so update it to emit the selected instructor.
23. In the parent component (`app.component`) we need to create a new method to handle this event. Firstly, create a new property in the component's class called `displayCourses`. Using a spread operator (or any other method you'd like) create a clone of the `courses` array. Now update the template so that the `displayCourses` array is used rather than `courses`.
24. Still in the component's class, create a new method called `doInstructorChange` which takes an `Instructor` as a parameter and returns a void. This method should use the `filter` method of the `courses` array to compare each course to the list of

courses in this instructor's courses property. You'll find both [filter](#) and [includes](#) methods useful here.

25. Now bind the [doInstructorsChange](#) handler to the [change](#) property of the instructors gallery. Don't forget to pass the \$event object.

If you have time

In this part of the exercise you are going to add the ability to specify how large the thumbnail icons in the instructor gallery should be at the time of use of the component through the application of an Input data binding.

26. Create an input property on the instructor gallery component called [thumbnailTShirtSize](#) with an alias of [thumbnails](#). This will receive "t-shirt size" input from the parent component (i.e. "s", "m" or "l" sizes)
27. In the app.component template bind to the [thumbnails](#) property and give it a value of "s" for now.
28. Within the instructor component class create a [thumbnailStyles](#) object which has a "width" property set to the string "75px" - this acts as our default size.
29. Use an appropriate lifecycle hook to take the t-shirt size passed into the component and convert it into pixel sizes (suggest: 75px, 100px, 150px) and override the string in thumbnailStyles.width - remember to cater for unknown values.
30. Within the instructor template add an ngStyles binding to the thumbnail image which points to the thumbnailStyles object.
31. You should now be able to specify in the parent component that you want either small, medium or large thumbnail images displayed.