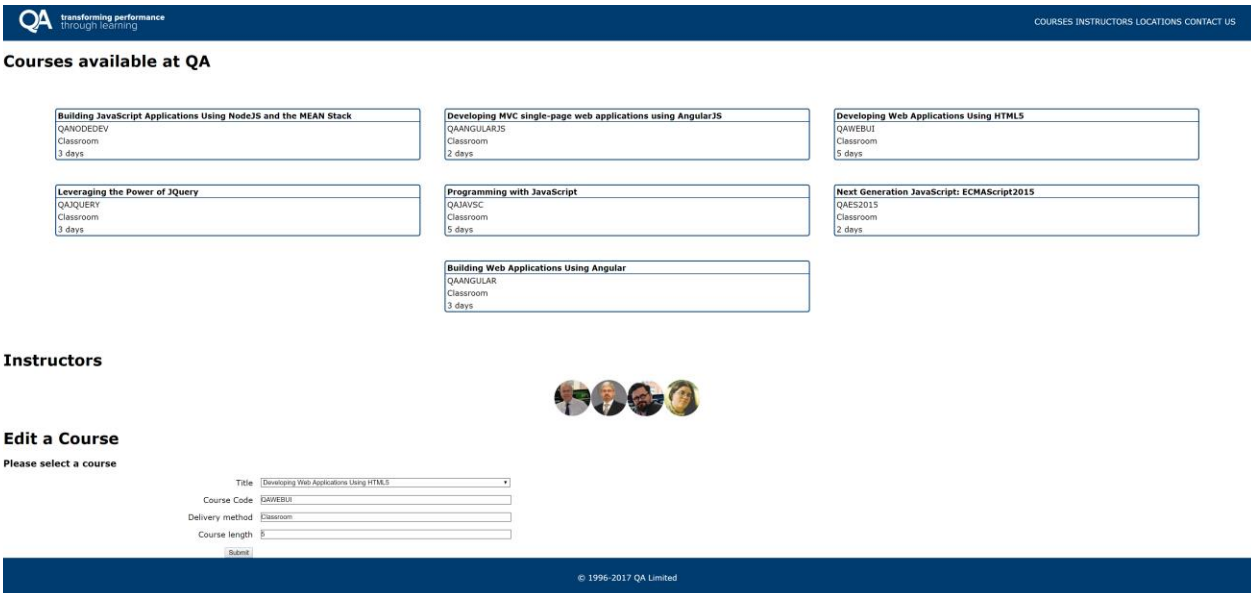
Exercise 8 - Creating Angular Forms

Objective

To create a new component within our Angular application that is able to receive input from the user through the use of a form.

Overview

In this exercise you are going to initially create a form which will contain very little "Angular" we will then furnish it with various Angular directives to provide feedback to the user as well as restrict their data entry to fit with our requirements. The end result should look something like this



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
QAWEBUI
Classroom
5 days

Leveraging the Power of JQuery
QAJQUERY
Classroom
3 days

Programming with JavaScript
QAJAVSC
Classroom
5 days

Next Generation JavaScript: ECMAScript2015
QAES2015
Classroom
2 days

Building Web Applications Using Angular
QAANGULAR
Classroom
3 days

Instructors

Edit a Course

Please select a course

Title	Developing Web Applications Using HTML5
Course Code	QAWEBUI
Delivery method	Classroom
Course length	5

Submit

© 1996-2017 QA Limited

Instructions

Part 1 - Creating a new feature module

First we need to set up our feature module. We're going to create the necessary files and then populate the model and the template so that we can view each course in a form, which can then be edited.

1. Create a new module called courses. Don't forget to import this module into the root module.
2. Add a new component to this module called course-editor.
3. Create a new model to represent a Course, it should have the following properties:

id	number
title	string
code	string
delivery	string
days	number
description	string

4. Define an array of Course objects within your new component and copy the courses array from the root module component into it. This isn't very DRY but that's a problem for another time.
5. Create a form with fields for each of the properties we defined above. The 'title' field should be a select element.
   Don't spend too long worrying about appearance - we're more concerned with Angular than we are with CSS right now.

Part 2 - Selecting a course for editing

Now that we have a form to display the courses we now need to bind it to the model such that when we select a course using the select element, its details are displayed in the appropriate fields of the form. Edits to the details in the form should be reflected in the model.

6. In the component class, add a `selectedCourse` property. This will hold the Course object when a user selects one via the select field.
7. We want the `selectedCourse` property to be set when the user changes the select field, to do this you should to two things:
   a. Use the ngModel directive to bind the select element to selectedCourse
   b. Use the ngValue directive to set the value of each option to be the course object from the ngFor like so:

```
<option *ngFor="let course of courses" [ngValue]="course">
    {{course.title}}
</option>
```

8. Now hook up two-way data bindings between the remaining input fields and the relevant properties of the `selectedCourse` object.

9. When the page first loads the `selectedCourse` property is undefined, and so your ngModel references will throw errors. To solve this wrap all of the inputs (not the select element!) in a div which uses ngIf to only display when `selectedCourse` exists.
10. Test the page. At this stage your form should be populating with course data whenever the user changes the select field.

Part 3 - Feeding back to the user

In this section we are going to utilise Angular's ngModel directive to provide feedback to the user as they complete the form.

11. Let's add a little validation to the form. Start by setting a template variable for the form using the ngForm directive export
12. Using an Angular property binding set the `disabled` property of the select element and submit button to be true while the form is invalid.
    a. Create a template variable on the form element which is assigned to ngForm
    b. The above template variable will have a `form` property which has an `invalid` boolean. Use this to set the disabled binding to be true/false as appropriate.
13. Add required attributes to the code, delivery and days fields.
14. Check that you can't change to a different course, or click the submit button should you leave one of the fields blank.
15. Let's add a simple red/green indicator to visually indicate what parts of the form are correct or incorrect.
    a. Create CSS classes for when the text fields have been edited and are then either valid or invalid.
    b. Set the right-side border to `8px solid red/green` for the invalid/valid classes.
16. Colour (especially red/green!) should never be used as the sole indicator. Add an error message to each field after the input box that only displays when the field has not been completed.
17. Finally, write a function to handle the submission of the form. For the time being it should simply log the `selectedCourse` to the console.

If you have time…Custom Validators

Create a custom validator that would ensure that the Course Code supplied by the user starts with the letters QA (or qa) and has at least 3 more letters after this.

The Regular Expression needed for this is:

```
[Qq]{1}[Aa]{1}[A-Z]{3,}
```

Within the code, we will ensure that this is case insensitive.

Start by creating a Custom Validator Directive:

18. Navigate to the **courses** folder and create a new **directive** called `allowed-course-codes` using the Angular-CLI
19. In the **allowed-course-codes.directives.ts** file, add an import for `Input` from `@angular/core`
20. Imports the following from `@angular/forms`:

- `AbstractControl`
- `NG_VALIDATORS`
- `Validator`
- `ValidatorFn`
- `Validators`

The Validator Function

21. Under the import statements, export and declare a function called `allowedCourseCodeValidator` that:

    - Takes a **RegExp** called `allowedCourseCodeRe` as an argument
    - Has a return type of `ValidatorFn`
    - Returns an anonymous function that:
        - Takes an **AbstractControl** called `control` as an argument
        - Has a return type `{[key: string]: any}`
        - Declares a **const** called `allowed` in the function body that is assigned to the result of **testing** the *supplied RegExp* against `control.value`
        - Returns the ternary evaluation of **'NOT'** `allowed` – this should return `{'allowedCourseCode': {value: control.value}}` or `null`

The Validator Decorator

22. Change the selector in the @Directive decorator so that it is `'[allowedCourseCode][ngModel]'`
23. Add `providers` that set:

```
[{provide: NG_VALIDATORS, useExisting: AllowedCourseCodeDirective,
multi: true}]
```

The Validator Class

24. Make the `AllowedCourseCodeDirective` class **implement** `Validator`
25. In the class body, remove the `constructor() {}` and add an `@Input()` decorator with `allowedCourseCode` as a **string**
26. Still in the class body, add a **method** called `validate` that:
    - Takes an **AbstractControl** called `control` as an argument
    - Has a return type `{[key: string]: any}`
    - Has a body that **returns** a ternary statement evaluating `this.allowedCourseCode` with `allowedCourseCodeValidator(new RegExp(this.allowedCourseCode, 'i'))(control)` or `null` as the return values

Modifying the template to use the validator

27. In the **input** for Course Code, add an attribute called `allowedCourseCode` and assign it to the **RegExp string** supplied at the start of the section
28. Replace the current `<div>` under the **input** with a `<div>` that uses the `*ngIf` directive to look for `code.invalid && (code.dirty || code.touched)`
    - Inside this `<div>` add another `<div>` that uses `*ngIf` to check `code.errors.required` and display a message that asks for a course code to be entered
    - Add a sibling `<div>` to the one added in the previous step that uses `*ngIf` to check `code.errors.allowedCourseCode && !code.errors.required` and display a message explaining that the course code is incorrect
29. Save all files and check your application – you should now see the appropriate validation message when the course code is removed or it is not in the correct format

If you really, really have time…

Eagle-eyed delegates may have spotted that there is an alternative way to do this using the built-in validators. Examine the Validators API and see if you can work it out…

Hint: It's to do with length and pattern…