



# Learn Angular

BY MAKING A QUIZ APP  
FROM SCRATCH

[HUNGRYTURTLECODE.COM](http://HUNGRYTURTLECODE.COM)

## Part 16



# Check Out The Whole Course Index

1. [Getting Started](#)
2. [Ng-controller directive and the \(mis\)use of \\$scope](#)
3. [Looping around with the ng-repeat directive](#)
4. [Markup for the Bootstrap Modal](#)
5. [Using Angular Filters to create real time search](#)
6. [The powerful ng-click directive](#)
7. [Services in Angular make everything easier](#)
8. [What is this infamous dependency injection in Angular?](#)
9. [Let's build a Factory](#)
10. [The ng-class directive](#)
11. [More Bootstrap markup – the well](#)
12. [Adding some logic to the controller](#)
13. [Making things disappear with ng-if](#)
14. [The \\$index property for ng-repeat](#)
15. [Reusing code is always a good idea](#)
16. You are here
17. [The final prompt after the quiz](#)
18. [Marking the quiz](#)
19. [More dependency injection](#)
20. [Reusing and slightly modifying some previous Bootstrap](#)
21. [More than one way to use ng-class](#)
22. [Another Angular Filter](#)
23. [More usage of Ng-if](#)
24. [Finishing The App](#)

## Errors Must Be Handled In All Apps

Handling errors is a critical part of the life of a software developer. If you cannot handle errors, you will create code that has a ton of bugs and just doesn't function satisfactorily. In this part of the series we will use [bootstrap alerts](#) to display an error message when the user tries to continue at the last question but they haven't answered all questions yet.

If you want to see the app for yourself, [check it out here](#).

The git repo [can be found here](#).



Click the image to go to the video on youtube or go here [https://www.youtube.com/watch?v=doU9zxZBoXI&list=PLqr0oBkln1FBmOjK24\\_B4y\\_VAA8736wPq&index=16](https://www.youtube.com/watch?v=doU9zxZBoXI&list=PLqr0oBkln1FBmOjK24_B4y_VAA8736wPq&index=16)

[The next part can be found here](#)

As I mentioned, the error should display when the user is on the last question and continues. If you remember back to the code in the setActiveQuestion function, we have some logic that increments the active question unless it is on the last

question, in which case it resets it to 0.

When it resets to 0, it means the user was on the last question and clicked continue without all questions being answered (if all questions had been answered, the setActiveQuestion function would never have been called). This is when we want to display the error.

Using this knowledge of the reset back to 0, we can create a conditional after the potential reset to 0 that checks if it has in fact been reset. If it has, then we display the error.

```
while(!breakOut){  
    vm.activeQuestion = vm.activeQuestion < quizLength?++vm.activeQuestion:0;  
    if(vm.activeQuestion === 0){  
        vm.error = true;  
    }  
    if(DataService.quizQuestions[vm.activeQuestion].selected === null){  
        breakOut = true;  
    }  
}
```

Notice inside the new conditional we reference **vm.error**. We have yet to create this, so add it at the top of the controller where we declare all the other properties. This what the whole quiz controller looks like so far:

```

(function(){
    angular
        .module("turtleFacts")
        .controller("quizCtrl", QuizController);

    QuizController.$inject = ['quizMetrics', 'DataService'];

    function QuizController(quizMetrics, DataService){

        var vm = this;

        vm.quizMetrics = quizMetrics;
        vm.dataService = DataService;
        vm.questionAnswered = questionAnswered;
        vm.setActiveQuestion = setActiveQuestion;
        vm.selectAnswer = selectAnswer;
        vm.activeQuestion = 0;
        vm.error = false;

        var numQuestionsAnswered = 0;

        function setActiveQuestion(index){

            if(index === undefined){
                var breakOut = false;

                var quizLength = DataService.quizQuestions.length - 1;
            }
        }
    }
}

```

```

        while(!breakOut){
            vm.activeQuestion = vm.activeQuestion < quizLength?++vm.activeQues
tion:0;

            if(vm.activeQuestion === 0){
                vm.error = true;
            }

            if(DataService.quizQuestions[vm.activeQuestion].selected === null)
{
                breakOut = true;
            }
        }else{
            vm.activeQuestion = index;
        }
    }

    function questionAnswered(){
        var quizLength = DataService.quizQuestions.length;

        if(DataService.quizQuestions[vm.activeQuestion].selected !== null){
            numQuestionsAnswered++;
            if(numQuestionsAnswered >= quizLength){
                // Finalise Quiz. Will tackle the code to put here later in the co
urse
            }
        }

        vm.setActiveQuestion();
    }

    function selectAnswer(index){
        DataService.quizQuestions[vm.activeQuestion].selected = index;
    }
}

})();

```



## Next, Create The Markup For The Bootstrap Alert

At the top of the row that holds the question area will be where we place the [bootstrap alert](#). Just above the `<h3>Question:</h3>`.

The screenshot shows a web application titled "Turtle Facts Quiz". Below the title is a subtitle: "Learn about all the turtles below before you decide to take on the **TURTLE QUIZ**".

Below the subtitle is a "Progress:" section with a row of 10 circular icons. The first 9 icons are red with a white question mark, and the 10th icon is blue with a white pencil. A red arrow points to the 10th icon, with the text "Bootstrap alert error message" written above it. To the right of the progress icons is a "Legend:" section with two items: "Answered" (blue square with a white pencil) and "Unanswered" (red square with a white question mark).

Below the progress and legend is a red alert box with the text "Error! You have not answered all of the questions!" and a close button (an 'x' icon) on the right.

Below the alert box is a "Question:" section. The first question is "1. How much can a loggerhead weigh?". There are four input fields with the following options: "Up to 20kg", "Up to 115kg", "Up to 220kg", and "Up to 500kg". At the bottom left of the question section is a "Continue" button.

The markup to create the alert box is simple. We just need to add the class of alert and the class of alert-danger to make it red. I also added a button with the class of close and the value of `&times` which will create a nice x to close the alert box.

On this button is an ng-click directive that will change the error property back to false which will remove the error box when we click the x button. Instead of calling a function from the ng-click like we have been doing throughout this course, it is also possible to just make an assignment declaration.

Obviously, we only want this alert box to display when the error property is true (then disappear when we hit the close button that sets error back to false). To do this we can use ng-show.

```
<div class="alert alert-danger"
      ng-show="quiz.error">

    Error! You have not answered all of the questions!

    <button class="close" ng-click="quiz.error = false">&times</button>

</div>
```

This works because `quiz.error` is already a boolean value, so as it changes it will show and hide the error box.

## Finalising The Quiz

You may remember earlier, inside the `questionAnswered` function we checked if the current question had been answered, then we checked if all questions had been answered. Inside the second conditional we said we would add code to finalise the quiz. This is what we need to do now. Here is the code to help your memory:

You may be wondering why we have the line **`numQuestionsAnswered >= quizLength`** rather than simply **`numQuestionsAnswered === quizLength`**. I mean, if all questions have been answered `numQuestionsAnswered` will be exactly equal to the length of the quiz. Right?

The problem is that we do not stop the user from clicking continue on the same answered question twice. For example, the user answers question 1 and clicks continue which will call the `questionAnswered` function and increment `numQuestionsAnswered` and move us to question two.



Now suppose that the user realises the answer they gave to question one was wrong. So they go back to question one, change their answer and click continue again. This will then call the `questionAnswered` function and increment `numQuestionsAnswered`.

This leaves us in a position where `numQuestionsAnswered` is 2 but the actual number of questions that have been answered is only 1. This is why we need to have a `>=` check.

This then leads us to a situation where the user may have `numQuestionsAnswered` greater than or equal to the length of the quiz but they still haven't answered all the questions. So we need to add a final sanity check that loops through all the questions just to make sure they are all answered.

```
if(numQuestionsAnswered >= quizLength){  
    for(var i = 0; i < quizLength; i++){  
        if(DataService.quizQuestions[i].selected === null){  
            setActiveQuestion(i);  
            return;  
        }  
    }  
    // More code here shortly  
}
```

We call `setActiveQuestion` and pass in `i` as an argument. We do this because we know that at index `i`, the question isn't answered, because the if statement triggered. So we don't need to waste computations trying to find the unanswered question, we can just pass it in.

When setActiveQuestion is finished, we return from the function. This is because if we didn't return, the function would keep running and get down to the line below that again calls setActiveQuestion. There is no need for this, so we just return from the function.

## All Checks Passed, Time To End The Quiz

If all questions have indeed been answered, then the if statement will never trigger and we won't call setActiveQuestion or return from the function, the code below the for loop will start to run. So this is where we add the code to do the housekeeping for the quiz before we move onto the results.

```
if(numQuestionsAnswered >= quizLength){  
    for(var i = 0; i < quizLength; i++){  
        if(DataService.quizQuestions[i].selected === null){  
            setActiveQuestion(i);  
            return;  
        }  
    }  
  
    vm.error = false;  
    vm.finalise = true;  
    return;  
}
```

The finalise property also needs to be initialised at the top of the controller, so don't forget to do that.

Using the finalise property that is set to true when all questions have been answered we can display a little prompt that asks

the user if they are sure they want to continue to the results, or if they would like to stay on the quiz to change some answers.

## **All Done Here, More In Part 17**

This is what we will tackle in the [next part](#). After that we will be able to create the third and final controller – the results controller.

See you in [part 17](#).

Adrian