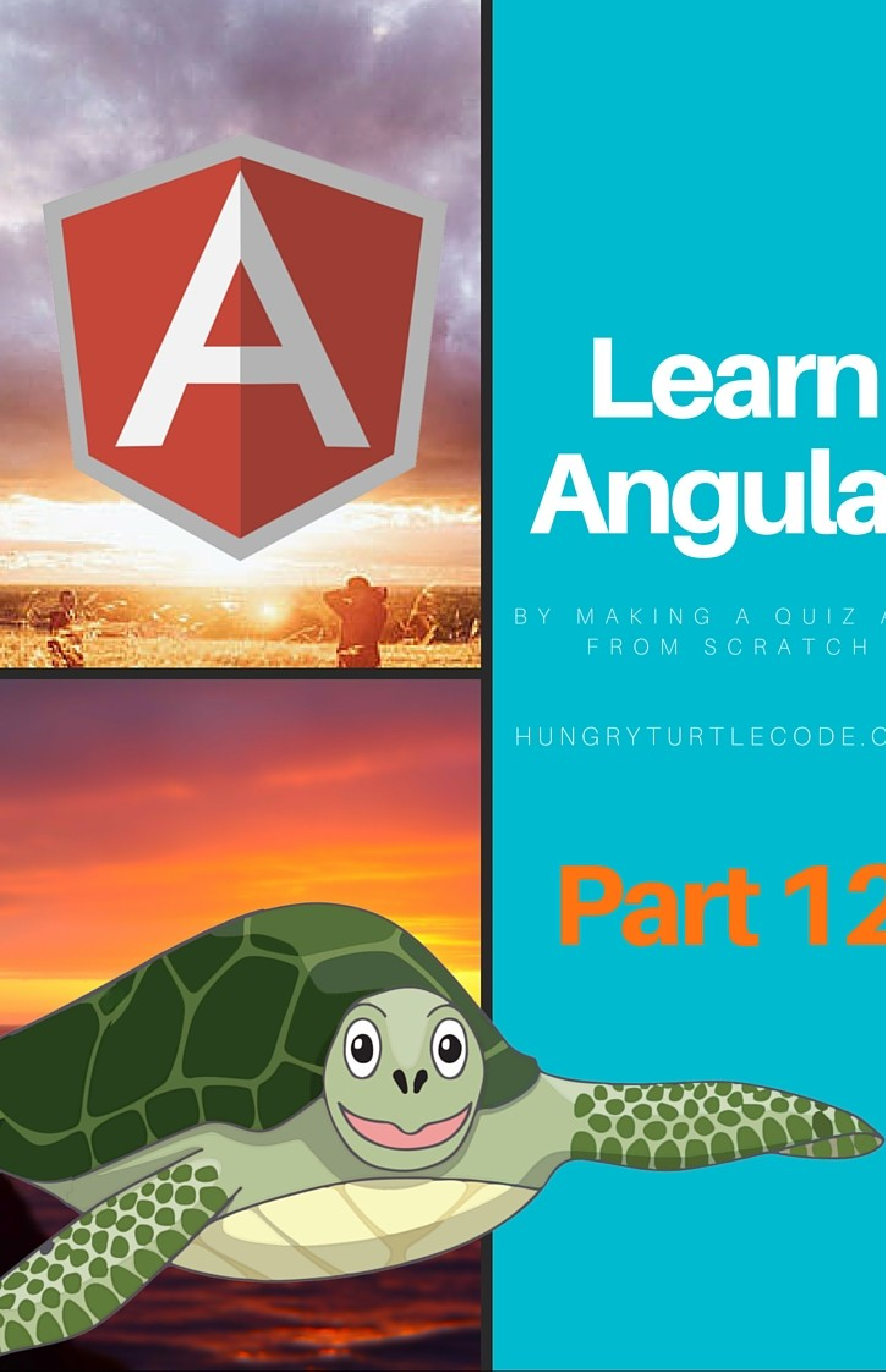


Learn Angular

BY MAKING A QUIZ APP FROM SCRATCH

HUNGRYTURTLECODE.COM

Part 12

Check Out The Whole Course Index

Controller Logic Keeps Us Going

We set ourselves up nicely in the last part. The ng-click calls a function on the quizController every time the “continue” button is clicked. This function will then need to change the activeQuestion property to the next available unanswered question in the quiz. So follow along with the article or the video as we build this controller logic.

If you want to see the app for yourself, check it out here.

The git repo can be found here.

Video Tutorial, For The More Visually Inclined

Click the image to go to the video on youtube or go here https://www.youtube.com/watch?v=XamHS-0PiMM&list=PLqr0oBkln1FBmOjK24_B4y_VAA8736wPq&index=12

The next part can be found here.

Let’s get started on the questionAnswered function to get this functionality working. So let’s first take an overview look at what this function will need to do.

For starters, we need to check if the question the user has just clicked continue on has been answered or if it has been skipped. If it has been answered we need to increment a variable that will count the total number of question that have been answered in the quiz so far. This will be used to allow us to know when the user has finished. We will obviously need to create this variable on the controller.

Once we have incremented the total questions answered, we check if the quiz has been finished ie. the total questions answered is equal to the total number of questions in the quiz.

If the quiz isn’t finished then we must increment the activeQuestion property to the next available unanswered question.

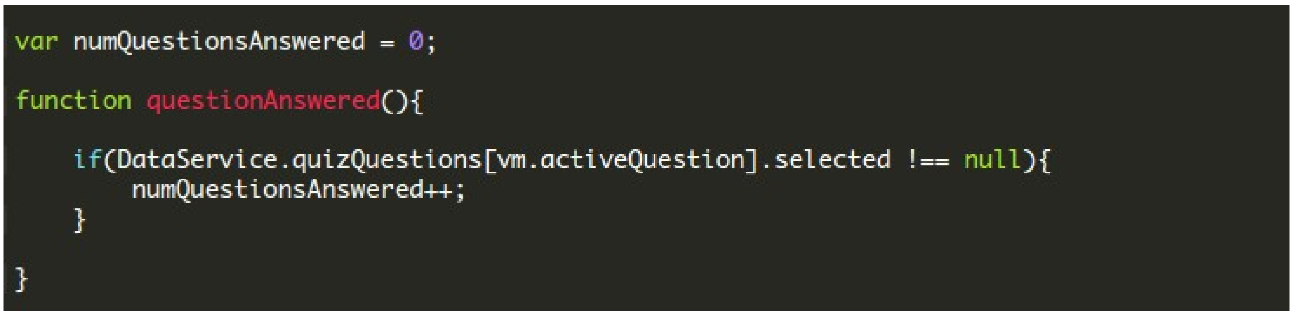
Controller Logic To Check If The Question Has Been Answered

If you remember back to the JSON for each question you will remember that there is a selected flag for each question. This is initially set to null, meaning the question has not been answered. As soon as the user does answer the question then this selected property will be set to the index of the possible answer they selected (0-3).

We still need to create the function that will set this selected value, but just keep in mind that this is something we will be doing in the future.

With this knowledge, it is now possible to detect if the question has been answered by checking if the selected property for this question equals null. If it does, the question hasn't been answered and if it does then, of course, it has been answered.

This is the start of the questionAnswered function:



```
var numQuestionsAnswered = 0;

function questionAnswered(){

    if(DataService.quizQuestions[vm.activeQuestion].selected !== null){
        numQuestionsAnswered++;
    }

}
```

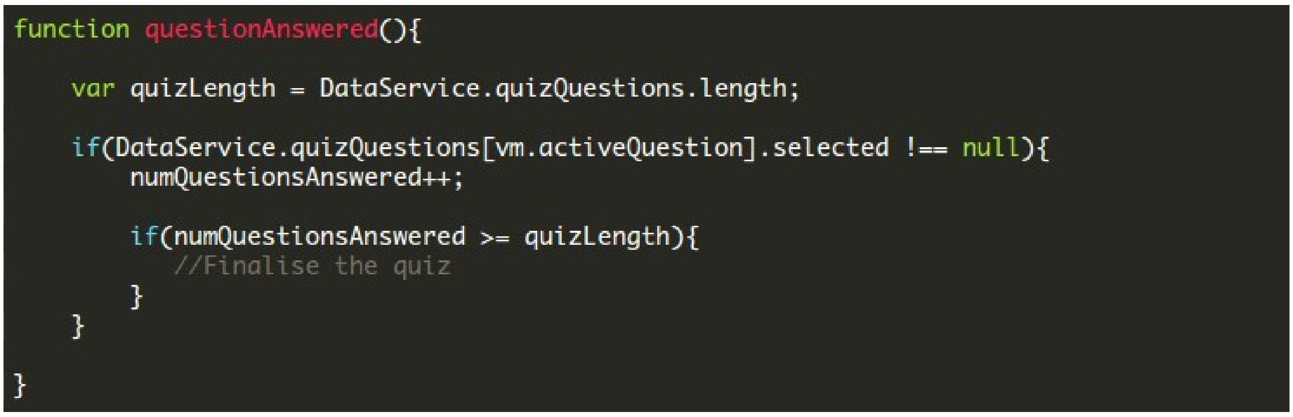
You can see that we initialise a numQuestionsAnswered variable. Notice that we used var and not vm. To create the variable. This is because the view does not need access to this variable. It is exclusively for the controller to use.

Then we check to see if the question has been answered, ie the selected property for this question doesn't equal null. If the question has been answered, then we increment the numQuestionsAnswered variable.

Check If All Questions In The Quiz Have Been Answered

Inside the conditional statement we incremented the number of questions answered. Now that we have done that, we don't want to move straight onto the rest of the function. Instead, we want to make sure the user hasn't just finished the quiz. There is no point doing all this extra logic if they have just answered the final question.

Still inside the conditional we can add a further conditional to check if we are at the end of the quiz. To check this we simply compare the number of answered questions to the total number of questions in the quiz.



```
function questionAnswered(){

    var quizLength = DataService.quizQuestions.length;

    if(DataService.quizQuestions[vm.activeQuestion].selected !== null){
        numQuestionsAnswered++;

        if(numQuestionsAnswered >= quizLength){
            //Finalise the quiz
        }
    }

}
```

Just to make things easier we stored the length of the quiz in a variable before the conditional, incase we need it again. The logic inside the second conditional is something we will return to later. But the logic will do some final checks and then return from the function call.

Now that the checks have been done the further logic that needs to be done when the quiz isn't finished can be added. At the end of the questionAnswered function, after the conditional blocks we can think about incrementing the activeQuestion to the next available unanswered question.

Avoiding Annoyances In Applications

This bit of code in our app isn't as simple as just incrementing activeQuestion. This is because we have the buttons in the progress bar that will allow the user to hop between questions.

For example, let’s say the users skips the first question without answering it. They are taken to question two. They then answer it and are taken to question three. At this point they remember the answer to question one, so they hit the button in the progress area to take them back to question one.

Hitting this button will change activeQuestion to the index of the first question, 0 (we will implement this logic soon). Now they answer the question and hit continue. If we only incremented the activeQuestion when the user hits continue, they will be taken to question two, which they have already answered.

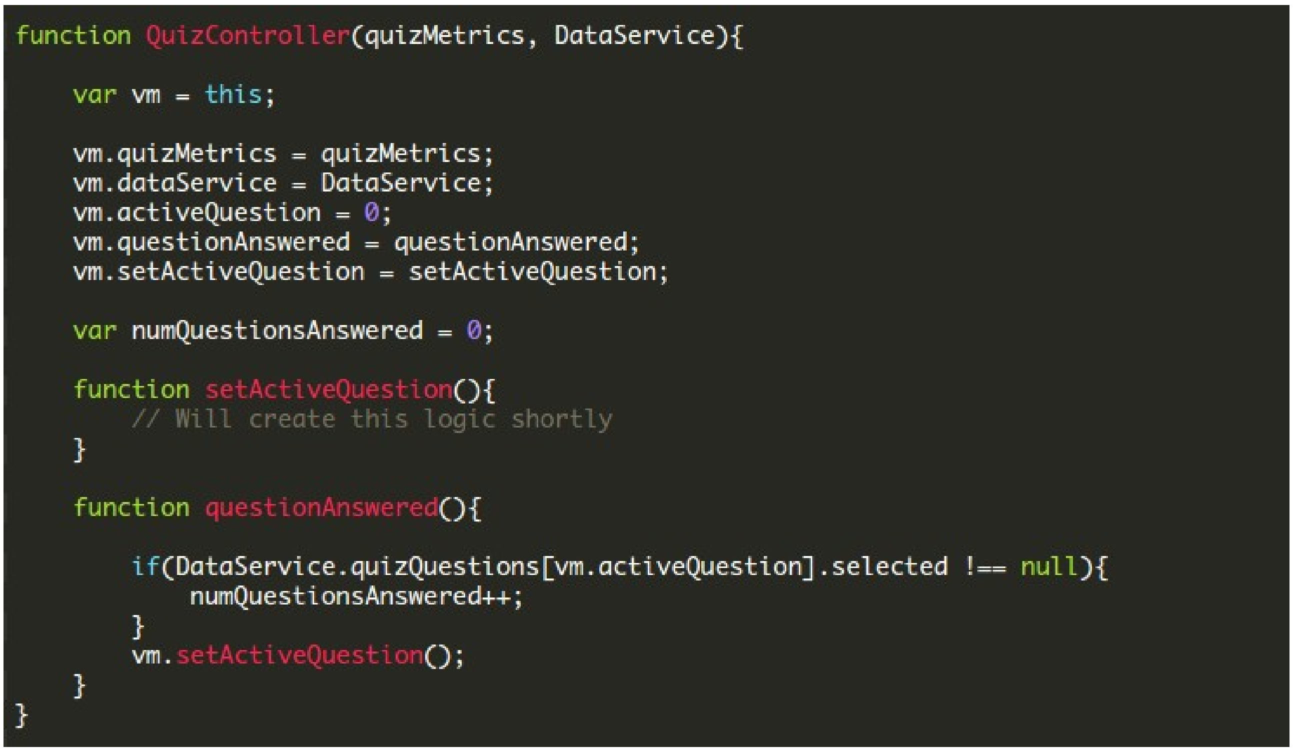
This is an annoyance, so I would rather avoid it.

Calling Another Function

Instead we need to scan through all the questions and find the lowest indexed question that has yet to be answered. In the previous example this logic would skip question two because it has been answered and it would instead find that question three is the lowest index that hasn’t been answered yet.

This is a bit more complex, so I would urge you to separate the logic into another function that we then simply call from the current function.

This is exactly what I have done and I will call the function setActiveQuestion. So I will create the function inside the controller just like we have been doing in the past and then call it from within the questionAnswered function using vm.setActiveQuestion().

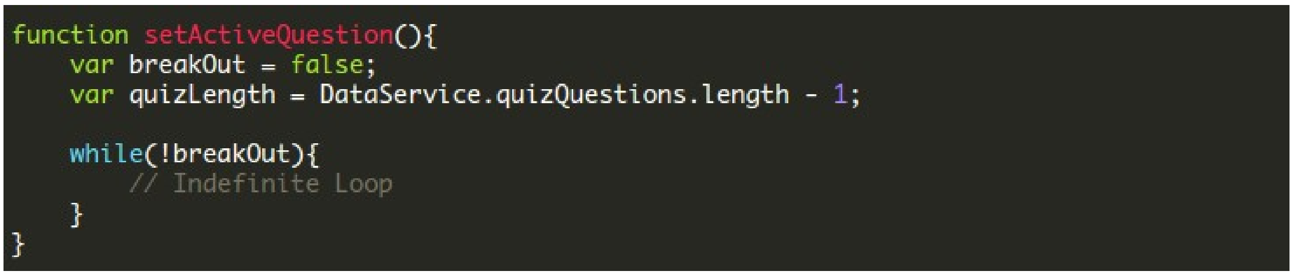


```
function QuizController(quizMetrics, DataService){

    var vm = this;

    vm.quizMetrics = quizMetrics;
    vm.dataService = DataService;
    vm.activeQuestion = 0;
    vm.questionAnswered = questionAnswered;
    vm.setActiveQuestion = setActiveQuestion;

    var numQuestionsAnswered = 0;

    function setActiveQuestion(){
        // Will create this logic shortly
    }

    function questionAnswered(){

        if(DataService.quizQuestions[vm.activeQuestion].selected !== null){
            numQuestionsAnswered++;
        }
        vm.setActiveQuestion();
    }
}
```

Here the setActiveQuestion function is obviously blank, but you can see how it was created and then called.

setActiveQuestion Is A Bit Loopy

Let’s now start building out the setActiveQuestion function. Here are the first few lines inside the function.

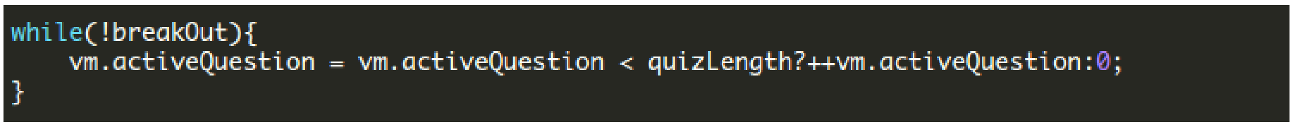


```
function setActiveQuestion(){
    var breakOut = false;
    var quizLength = DataService.quizQuestions.length - 1;

    while(!breakOut){
        // Indefinite Loop
    }
}
```

We are using a while loop here to loop indefinitely until we find a question that has yet to be answered. So we use a variable called breakOut, which is set to false and loop continuously, while breakOut is false. As soon as we find an unanswered question we will set breakOut to true, which will break the while loop and all is good.

The quizLength variable is pretty self explanatory as it is very similar to the one we created earlier. The main difference this time is that we are taking 1 away from the length. We do this because activeQuestion is 0 index. So if the length of the quiz is 10 questions, the activeQuestion will cycle from 0-9. Hence we take one away from the actual length to make it 0 index too.

Inside the while loop we will add a kind of fancy looking line of code, but it achieving a simple task.



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

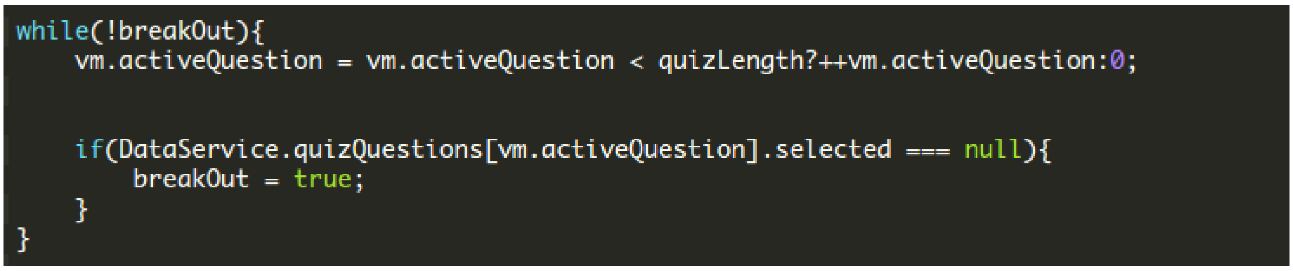
This is basically a condensed if statement. It checks if the activeQuestion is less than the length of the quiz. If it is, it simply increments activeQuestion and if activeQuestion isn't less than the length ie, it is the same size (last question in quiz) then we set activeQuestion back to 0 to continue looping from the start.

The reason we want this line of code to increment or reset activeQuestion to zero is simple. Say we skip question 1, but then answer all other questions. When we click continue on the final question, the code will check if all questions have been answered, which they havent been. This will then trigger the setActiveQuestion function.

If we only increment activeQuestion here, we will end up incrementing it past the total length of the quiz. Which would be invalid. So instead, we want to go back to the start of the quiz to find where that unanswered question is.

We know there is an unanswered question because we checked if total number of questions answered was equal to total questions and it wasn't. So we know there is an unanswered question. So go back and keep going until we find it.

Now we need some logic that will check if the current activeQuestion, after that last line (the incremented version or the reset back to zero) is unanswered.



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


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

Hopefully, this conditional make a bit of sense to you. We are simply testing to see if the question is unanswered, if it isn't then the while loop loops again and we again increment activeQuestion or reset it to zero if we where on the last question.

If the question is unanswered then the code inside the conditional runs and we set breakOut to true, which breaks the loop and ultimately returns the function with the newly found unanswered question as the active question.

Moving On To Part 13

In the next part we will fix some of the issues that you may have seen while flicking through the quiz in it’s current state – The image urls display instead of the images themselves for image based questions.

See you over in part 13

Adrian