
Aoife Pearson

Project Write up

Quiz Manager

1st July 2019 - 5th July 2019

OVERVIEW

This will be my ongoing Documentation for work completed from day to day. Here I will post a detailed journal of what I have completed each day throughout this project.

1st July

Requirements Review

I reviewed the new Requirements Document for the Quiz Manager project and considered the changes made and how they would affect my project. To begin with, I decided to follow the same process I had done to create my initial documentation, and make changes accordingly.

Feasibility Study

For the most part, the requirements had not changed dramatically. The most notable changes were to the naming conventions of the 'Edit', 'View', 'Restricted' Users. Additionally, the users were now to be Pre-compiled via CSV upload rather than the User Management system via an Accounts page in the website managed by the 'Edit' Users. This change made me alter my documentation to include these differences.

Requirements Document

The Requirements document needed to change more than I was expecting. I had put some planning steps in regards to the User Management System which would no longer be required. I had planned as part of future development to include the functionality to edit and delete user accounts if the User has edit rights, however, now that the users were pre-compiled this was no longer a necessary step.

User Stories

I had to rename my user roles throughout the User stories I had created and remove a couple which related to Users being able to add/edit/delete other users accounts. I did also add the new user story to Pre-compile user accounts from a CSV.

Flow Chart

I did not have to change my user flow chart as I had not originally added the User account management during the preparation phase of the project. Therefore my flow chart was still accurate as to the user's journey throughout my application.

WireFrame Designing

My Designs did not need to change, for the most part, I removed the now redundant designs that were created for the user management system but as per the flow chart, the user journey had not changed and I felt the views were compatible with the requirement for a diverse brand update for the site.

All CSS changes that affect the brand of the site should be controlled via a single CSS document, therefore the document can be “swapped” out for those of a different brand without affecting the site.

Testing

My test cases did not need to change as they followed the original user journey I had designed, which I still plan to implement, I did remove the test cases associated with the User management system.

GitHub

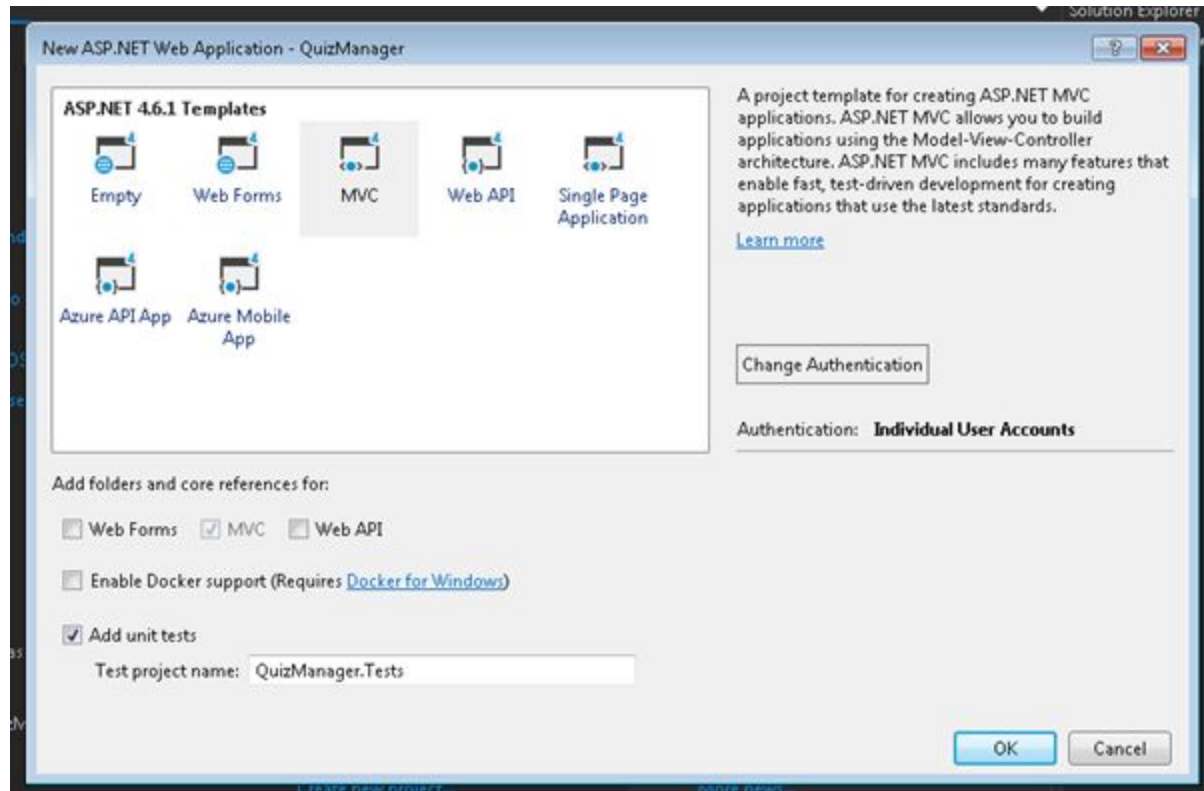
Once my user documentation was in order, I created a new [GitHub repository](#) to save my documentation and code to. I created a README.md and added snippets from the Brief to describe what I was expecting from the project.

I uploaded all my documentation up to this point as well which I will continue to do throughout this project.

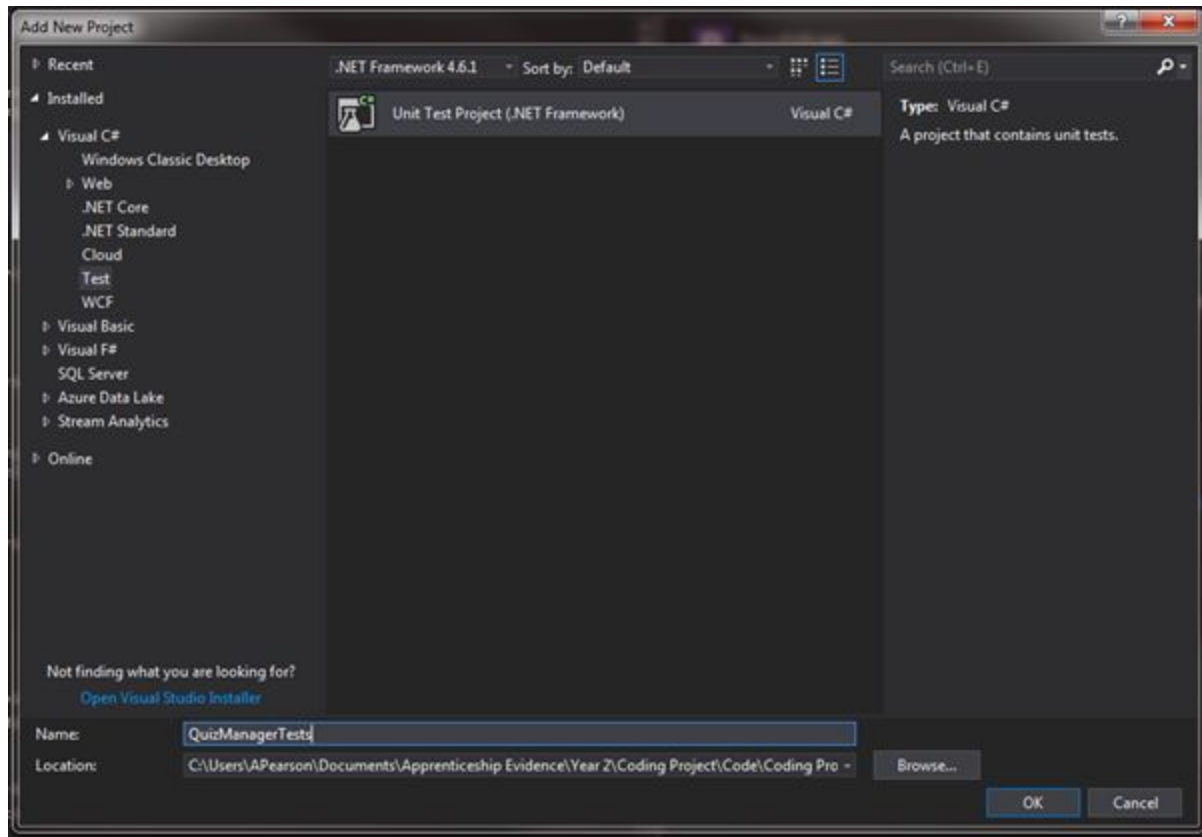
Project Creation

Once I had a repository setup I began to create my new project. My first step was to create a new Project inside my GitHub repo folder, I created an ASP.NET Web Application and selected the

MVC option. By selecting the MVC option I was able to make use of the Controller, Model, View components and would have the basic pages precompiled for me as well as all necessary libraries such as bootstrap and System. Web packages. I changed the Authentication mode to individual User Accounts, this would precompile the default Owin Identity code that would be used to manage users from a DB and set up the login/logout functionality. I also selected the Add Unit Tests so that I could create unit tests associated with my project.



After the Project was created I had to create the unit test project that would link to the Main Web Application.



Next, I created the DB using the attached script:

```
CREATE DATABASE QuizManager;
```

```
CREATE TABLE Category(  
    Id INT NOT NULL PRIMARY KEY,  
    Title VARCHAR(50) NOT NULL  
);
```

```
CREATE TABLE Quiz(  
    Id INT NOT NULL PRIMARY KEY,  
    DateCreated DATETIME NOT NULL,  
    Title VARCHAR(200) NOT NULL,
```

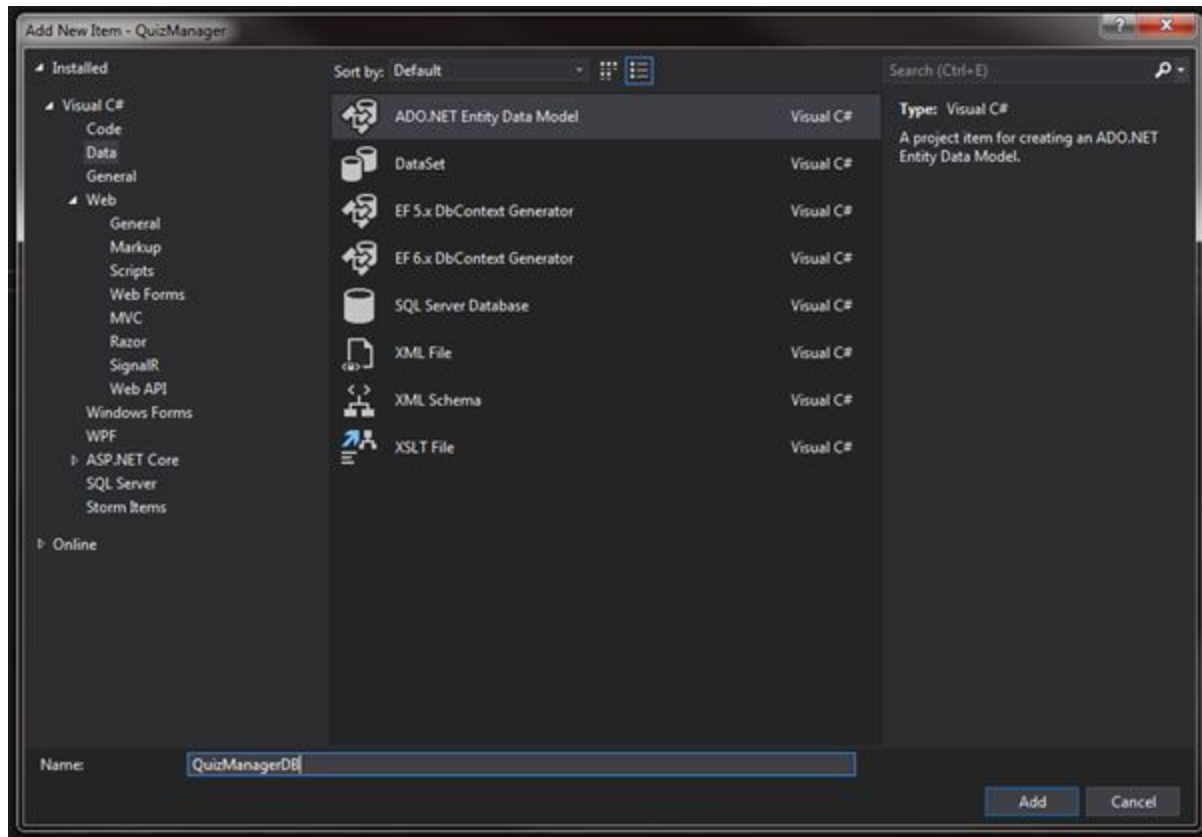
```
Description VARCHAR(300),  
Author VARCHAR(50) NOT NULL,  
PassMark INT NOT NULL,  
CategoryId INT FOREIGN KEY REFERENCES Category(Id) NOT NULL  
);
```

```
CREATE TABLE Answer (  
Id INT NOT NULL PRIMARY KEY,  
QuestionId INT FOREIGN KEY REFERENCES Question(Id) NOT NULL,  
Correct INT NOT NULL,  
Answer VARCHAR(300) NOT NULL,  
Explanation VARCHAR(1000) NOT NULL  
);
```

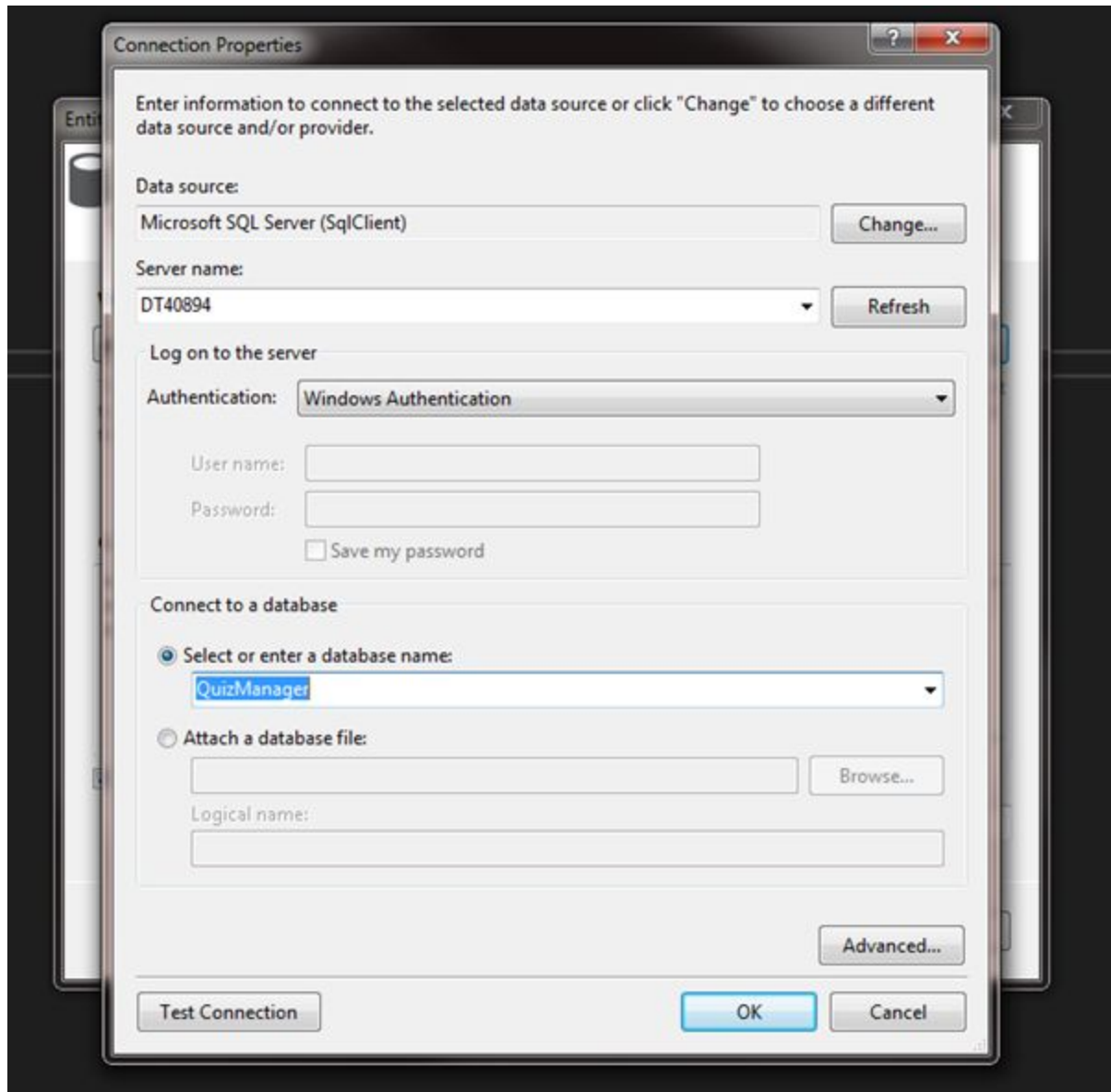
```
CREATE TABLE Question(  
Id INT NOT NULL PRIMARY KEY,  
QuizId INT FOREIGN KEY REFERENCES Quiz(Id) NOT NULL,  
Question VARCHAR(250) NOT NULL,  
);
```

I then Manually added the first Test Data using Insert statements so that my site would have some data to use while I was developing the site.

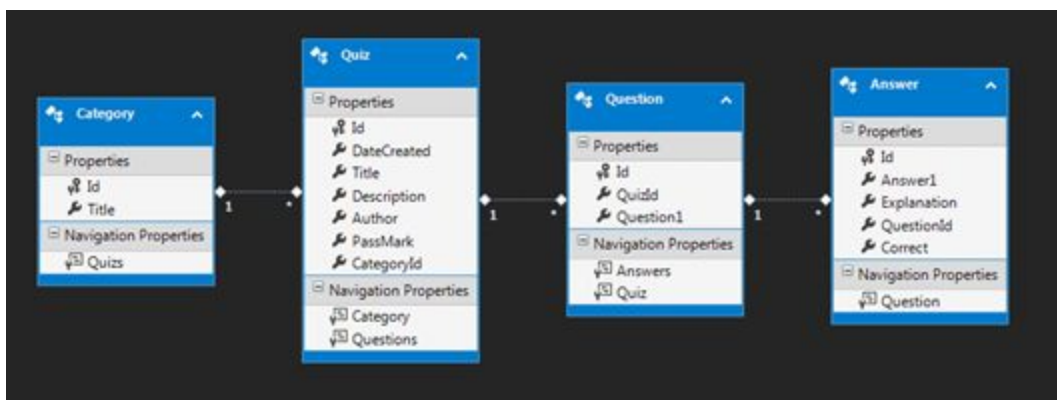
Now that I had a functional DB, I could implement it into my code. By using ADO.NET Entity Data Models I was able to use a DB first approach to connect my site to the DB and create the relevant classes automatically once the site had been loaded.



To Create an Entity Data Model, I first had to create the New item inside my project and name it something appropriate. Then I followed the setup steps for an EF Designer from database Model, which would create each table as its own class with related properties. It would also create a visual designer window so that I can see what relationships my tables will have to one another, many to many, one to many or one to one etc.



Here I connected my Entity Data Model to my existing DB hosted on my local machine.



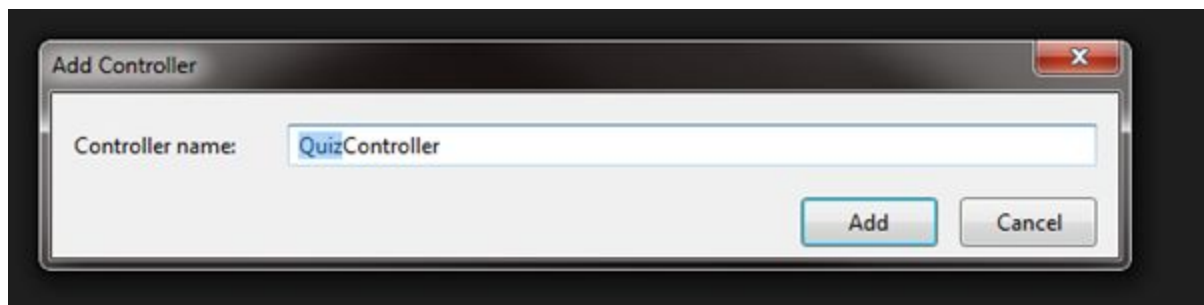
This was my final screen that showed my DB. The tables all have a 1 to many relationships and use foreign keys to reference one another.

CodeBase Set up

The next step was to set up my code structure so that I could implement my code throughout the project in a clear and concise manner. I decided that the files I would need to add were:

- QuizController
- QuizAdminController
- QuizViewModel
- TakeQuizViewModel
- QuizManagement > QuizAdmin

I created the Quiz Controller to enable the views to be created for each page relating to the User being able to take a quiz. I Also Created the QuizAdminController to manage the pages relating to Edit users being able to add or edit quizzes.



I would create the ActionResult once the appropriate Views had been set up first.

QuizViewModel

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5
6  namespace QuizManager.Models
7  {
8      8 references | AoifePearson@DSTOUPUT.co.uk, 18 hours ago | 1 author, 1 change
9      public class QuizViewModel
10     {
11         private QuizManagerEntities QuizEntities = new QuizManagerEntities();
12
13         2 references | AoifePearson@DSTOUPUT.co.uk, 18 hours ago | 1 author, 1 change
14         public QuizViewModel()
15         {
16         }
17
18         2 references | 0/2 passing | AoifePearson@DSTOUPUT.co.uk, 18 hours ago | 1 author, 1 change
19         public List<Quiz> GetAllAvailableQuizzes()
20         {
21             List<Quiz> placeholder = new List<Quiz>();
22             return placeholder;
23         }
24
25         1 reference | 0/1 passing | AoifePearson@DSTOUPUT.co.uk, 18 hours ago | 1 author, 1 change
26         public Quiz GetQuizByTitle(string quizTitle)
27         {
28             Quiz placeholder = new Quiz();
29             return placeholder;
30         }
31     }
32 }
```

The Quiz View Model will be used for retrieving quiz related data from the DB passing that to the Controllers and Views. The private QuizManagerEntities is used to connect to the DB making the generated classes accessible to this ViewModel.

TakeQuizViewModel

```
6 references | AoifePearson@DSTOUPUT.co.uk, 18 hours ago | 1 author, 1 change
public class TakeQuizViewModel
{
    //TakeQuiz Design:
    //Get the Full List of Questions Related to a Quiz by the Quiz Title
    //Add all Questions to AvailableQuestionsByNumber. code will check whether the question has been taken by the Key in the Dictionary.
    //GetQuestion() will generate a random question from the list of available questions
    //RemoveQuestionFromAvailableList() will ensure the question cannot be taken a second time
    //StoreQuestionToUsersAnswers() will keep a record of what answers the user provides
    //CheckUsersAnswers() will check how many question sthe user answered correctly and set the users final
    //score to "Pass" or "Fail" with their pass percentage as a string.

    private QuizManagerEntities QuizEntities = new QuizManagerEntities();
    1 reference | 0/1 passing | AoifePearson@DSTOUPUT.co.uk, 18 hours ago | 1 author, 1 change
    public Quiz CurrentQuiz { get; set; }
    7 references | 0/5 passing | AoifePearson@DSTOUPUT.co.uk, 18 hours ago | 1 author, 1 change
    public Dictionary<int,Question> AvailableQuestionsByNumber { get; set; }
    3 references | 0/3 passing | AoifePearson@DSTOUPUT.co.uk, 18 hours ago | 1 author, 1 change
    public Dictionary<Question,Answer> UsersAnswers { get; set; }
    2 references | 0/2 passing | AoifePearson@DSTOUPUT.co.uk, 18 hours ago | 1 author, 1 change
    public string FinalScore { get; set; }
    1 reference | AoifePearson@DSTOUPUT.co.uk, 18 hours ago | 1 author, 1 change
    public TakeQuizViewModel()
    {
    }
    6 references | 0/6 passing | AoifePearson@DSTOUPUT.co.uk, 18 hours ago | 1 author, 1 change
    public List<Question> GetListOfQuestionsByQuiz(string quizTitle)
    {
        List<Question> placeholder = new List<Question>();
        return placeholder;
    }
    1 reference | 0/1 passing | 0 changes | 0 authors, 0 changes
    public Question GetRandomQuestion()
    {
        Question placeholder = new Question();
        return placeholder;
    }
    1 reference | 0/1 passing | AoifePearson@DSTOUPUT.co.uk, 18 hours ago | 1 author, 1 change
    public void RemoveQuestionFromAvailableList(Question question)
    {
    }
    3 references | 0/3 passing | AoifePearson@DSTOUPUT.co.uk, 18 hours ago | 1 author, 1 change
    public void StoreQuestionToUsersAnswers(Question question)
    {
    }
    2 references | 0/2 passing | AoifePearson@DSTOUPUT.co.uk, 18 hours ago | 1 author, 1 change
    public void CheckUsersAnswers(Dictionary<Question,Answer> usersAnswers)
    {
    }
}
```

In the TakeQuizViewModel, I will be creating the code relating to the user taking a quiz.

Properties include:

- CurrentQuiz
 - To store the Current Quiz information throughout the use of the Quiz
- AvailableQuestionsByNumber
 - I plan to add a number to each Question using the Key of the dictionary so that questions can be randomly generated from the remaining questions not taken by the user
- UsersAnswers

- Here I plan to store the Q&A Pairs the User provides when they answer a question.
- FinalScore
 - Here I will use a string.format() to create the final pass/fail score that will be presented to the User

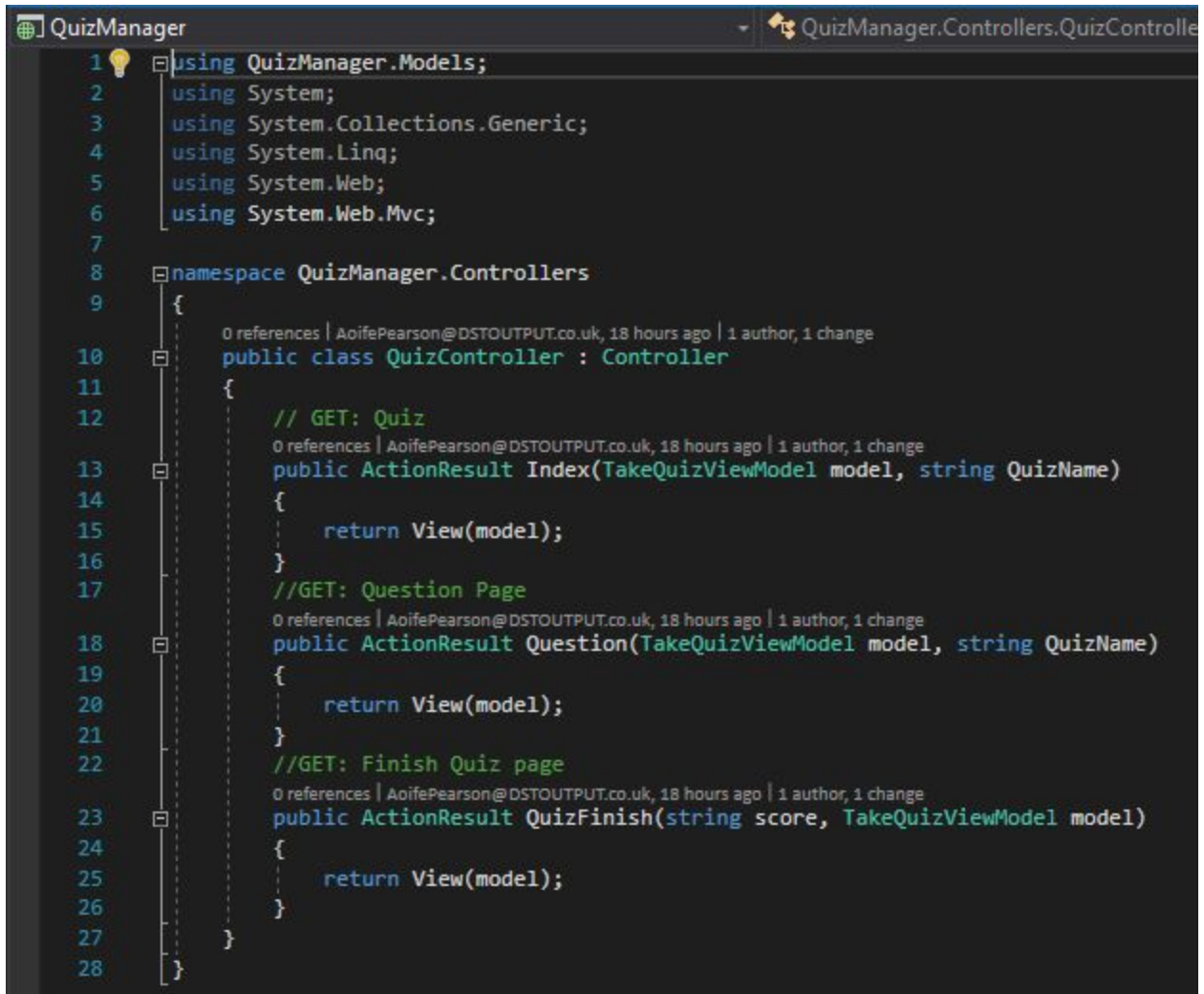
QuizAdmin.cs

```
QuizManager QuizManager.QuizManagement.QuizAdmin
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Web;
5
6 namespace QuizManager.QuizManagement
7 {
8     public class QuizAdmin
9     {
10         //Code behind Manage Quiz Page
11         private QuizManagerEntities QuizEntities = new QuizManagerEntities();
12         public QuizAdmin()
13         {
14         }
15
16         public void AddQuiz(Quiz newQuizItem)
17         {
18         }
19
20         public void AddCategory(string categoryName)
21         {
22         }
23
24         public void AddQuestionAndAnswers(Question question, List<Answer> answers)
25         {
26         }
27     }
28 }
29
```

This class was created to manage the Admin functionality for edit users, enabling them to add/edit a quiz.

I decided to only map out the Add functionality, to begin with, and then try to map out the Edit functionality once I had a strong code that I could base my “edit” work on.

QuizController



```
1  using QuizManager.Models;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Web;
6  using System.Web.Mvc;
7
8  namespace QuizManager.Controllers
9  {
10     0 references | AoifePearson@DSTOUPUT.co.uk, 18 hours ago | 1 author, 1 change
11     public class QuizController : Controller
12     {
13         // GET: Quiz
14         0 references | AoifePearson@DSTOUPUT.co.uk, 18 hours ago | 1 author, 1 change
15         public ActionResult Index(TakeQuizViewModel model, string QuizName)
16         {
17             return View(model);
18         }
19         //GET: Question Page
20         0 references | AoifePearson@DSTOUPUT.co.uk, 18 hours ago | 1 author, 1 change
21         public ActionResult Question(TakeQuizViewModel model, string QuizName)
22         {
23             return View(model);
24         }
25         //GET: Finish Quiz page
26         0 references | AoifePearson@DSTOUPUT.co.uk, 18 hours ago | 1 author, 1 change
27         public ActionResult QuizFinish(string score, TakeQuizViewModel model)
28         {
29             return View(model);
30         }
31     }
```

The Index View of my Quiz Controller will be my AboutQuiz Page, where the user will be able to view data about their chosen quiz as well as the button to start the quiz.

The Question View of my Quiz Controller will use the models `GetRandomQuestion()` to generate the view of the random question that was selected. This view will be refreshed each time the user clicks “next question”.

The QuizFinish Page will display once the user has clicked the “Finish Quiz” button. This includes the score, percentage correct and, if the User is an Edit, or View User, the Q&A pairs.

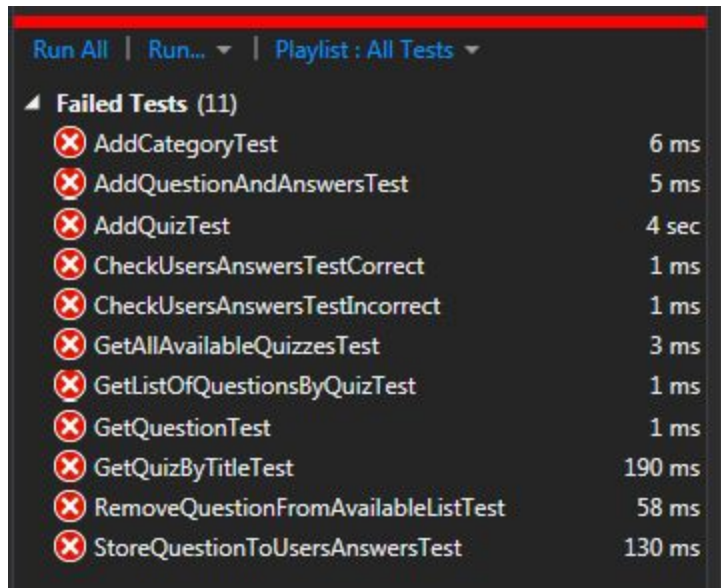
QuizAdminController

```
QuizManager
QuizManager.Controllers

1 using QuizManager.Models;
2 using System;
3 using System.Collections.Generic;
4 using System.Linq;
5 using System.Web;
6 using System.Web.Mvc;
7
8 namespace QuizManager.Controllers
9 {
10     0 references | AoifePearson@DSTOUTPUT.co.uk, 19 hours ago | 1 author, 1 change
11     public class QuizAdminController : Controller
12     {
13         // GET: QuizAdmin
14         0 references | AoifePearson@DSTOUTPUT.co.uk, 19 hours ago | 1 author, 1 change
15         public ActionResult Index()
16         {
17             return View();
18         }
19         //GET: Add Quiz
20         0 references | AoifePearson@DSTOUTPUT.co.uk, 19 hours ago | 1 author, 1 change
21         public ActionResult AddQuiz(QuizViewModel model)
22         {
23             return View(model);
24         }
25         //GET: Add Questions
26         0 references | AoifePearson@DSTOUTPUT.co.uk, 19 hours ago | 1 author, 1 change
27         public ActionResult AddQuestions(QuizViewModel model)
28         {
29             return View(model);
30         }
31         // GET: Add Category
32         0 references | AoifePearson@DSTOUTPUT.co.uk, 19 hours ago | 1 author, 1 change
33         public ActionResult AddCategory(QuizViewModel model)
34         {
35             return RedirectToAction("AddQuiz");
36         }
37         0 references | AoifePearson@DSTOUTPUT.co.uk, 19 hours ago | 1 author, 1 change
38         public ActionResult FinishQuizCreation()
39         {
40             return RedirectToAction("Index");
41         }
42     }
43 }
```

This view will control the Admin(Edit) pages of the site.

Unit Tests



I created my unit tests to test how I expected the methods to work. They all fail at the moment because of the methods only having placeholder code. But now they have been set up I can work on each method until all my tests pass.

Pre-compile Users from CSV

I first created a CSV File which contained 3 users I wanted to upload to the DB. my first step was to find where the Login and Register code was stored which was generated by visual studio when I clicked the “Individual User Accounts” option for Authentication on project startup. Both were found in the AccountController, so I moved the Register code to the Login ActionResult and made the ActionResult Async so that the Asynchronous method for creating a new user could run correctly.

I updated the RegisterViewModel to ensure the Role was added as a property, and then created the PrecompileUsers class.

PrecompileUsers.cs

```
2 references | 0 changes | 0 authors, 0 changes
public class PrecompileUsers
{
    public List<RegisterViewModel> Users = new List<RegisterViewModel>();
    1 reference | 0 changes | 0 authors, 0 changes
    public IEnumerable<string> GetAllUsers()
    {
        FileStream file = new FileStream("C:/Users/APearson/Documents/Apprenticeship Evidence/Year 2/Coding Project/Devel
        StreamReader streamReader = new StreamReader(file);
        string headerLine = streamReader.ReadLine();
        string line;
        while ((line = streamReader.ReadLine()) != null)
        {
            yield return line;
        }
        streamReader.Close();
    }
    1 reference | 0 changes | 0 authors, 0 changes
    public void ConvertCSVToUsers()
    {
        var csvLines = GetAllUsers();
        foreach (string line in csvLines)
        {
            string[] userProperties = line.Split(',');
            RegisterViewModel user = new RegisterViewModel()
            {
                Email = userProperties[0],
                Role = userProperties[1],
                Password = userProperties[2]
            };
            Users.Add(user);
        }
    }
}
```

I used this class to create a List<RegisterViewModel> so that I could create a user from each instance, then wrote a method GetAllUsers() to create an IEnumerable<string> from the CSV File. this output my CSV into lines, formatted as "Email, Role, Password" for each user. Each line became a string in the List.

Now I could create the ConvertCSVToUsers() which would be the only method called by the Login Action Result. This method retrieves the csv lines to a variable and iterates through them, splitting the csv by ',' character and then assigning the properties to an array. I then mapped those properties to the RegisterViewModel and added that ViewModel to the list of Users.

I added a Foreach loop in the Login ActionResult which would iterate through the list of Users in the precompileUsers class and register them to the DB.



ID	Email	EmailConfirmed	PasswordHash	SecurityStamp	PhoneNumber	PhoneNumberConfirmed	TwoFactorEnabled	LockoutEnabled
1	4722eb0-eb52-462b-b751-c52b335e0526	RestrictedUser@teat.com	0	AAnwZ0WUzaK3tr3WV8BcChTwtH2zE0CzCaCz3TpdFot4Tt4V	2574b7a7eab74117-8635-3da17111decc	NULL	0	NULL
2	889-8a65-7a30-47a7e43b-6c393c55a075	ViewUser@teat.com	0	ANagrGruZJwPUL8TW7acc57ePgeVf9baQ8a7p2ZVuaN7y8DV	14748a58a57c-4360-ad9e2b95627babb	NULL	0	NULL
3	c284F79-8e3-4b7e031-268b-7ec3a7c	EdtUser@teat.com	0	AQ55C0B0WwvF9Uwcu0aCeQ0e-FT2aHwA5-9B8Gh411YGLW	74b29e7b-4074-4558-acc0-78531795c777	NULL	0	NULL

Passwords were all hashed on upload.

Now that the users were precompiled I could remove the unnecessary precompile code and return the Login ActionResult to its previous state:

```
//  
// GET: /Account/Login  
[AllowAnonymous]  
0 references | AoifePearson@DSTOUPUT.co.uk, 5 hours ago | 1 author, 1 change  
public ActionResult Login(string returnUrl)  
{  
    //RegisterViewModel registerViewModel = new RegisterViewModel();  
    //PrecompileUsers precompileUsers = new PrecompileUsers();  
    //precompileUsers.ConvertCSVToUsers();  
    //foreach (RegisterViewModel userDetails in precompileUsers.Users)  
    //{  
    //    if (ModelState.IsValid)  
    //    {  
        var user = new ApplicationUser { UserName = userDetails.Email, Email = userDetails.Email };  
        var result = await UserManager.CreateAsync(user, userDetails.Password);  
  
        await UserManager.AddToRoleAsync(user.Id, userDetails.Role);  
  
    }  
    }  
    return View();  
}
```

I then decided to go through and remove all the unnecessary generated code by MVC for the project leaving only the Login/Logoff ActionResults and the LoginViewModel.

I kept the RegisterViewModel and PrecompileUsers classes as a reference in case more users were added in the future.

Log in/ Log out

Now that my users had been precompiled I ensured the login view was set up with only the relevant fields and tested my application to see if it would run.

Log in.

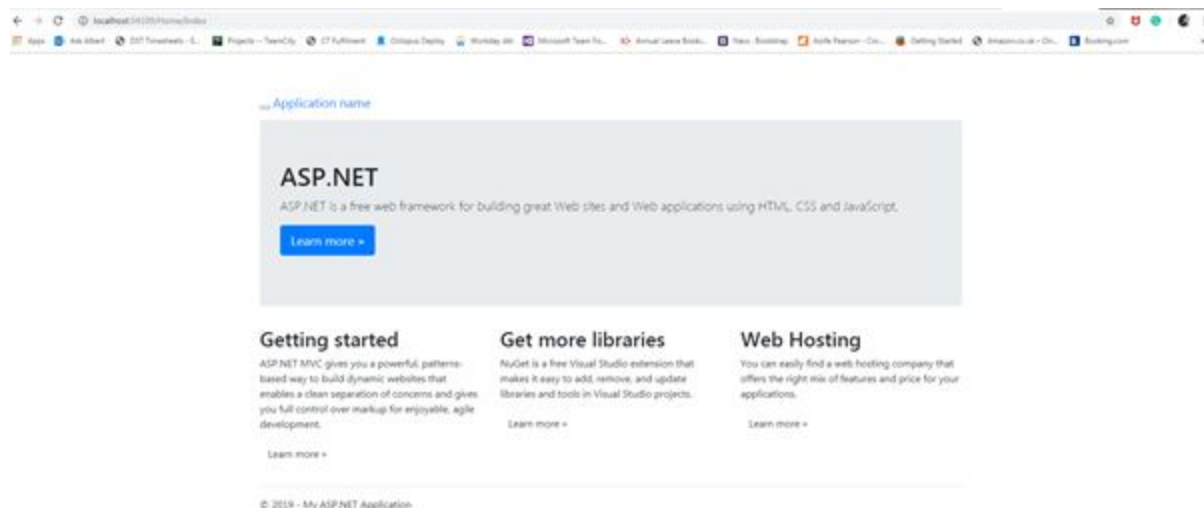
Use a local account to log in.

Email

Password

Log in

The Login screen could be navigated to, and when the user details that were precompiled were entered...



The users were able to login to see the default home page!

2nd July

QuizViewModel

I began the day by writing the code necessary for passing both my GetAllAvailableQuizzes Test and GetQuizByTitle Test.

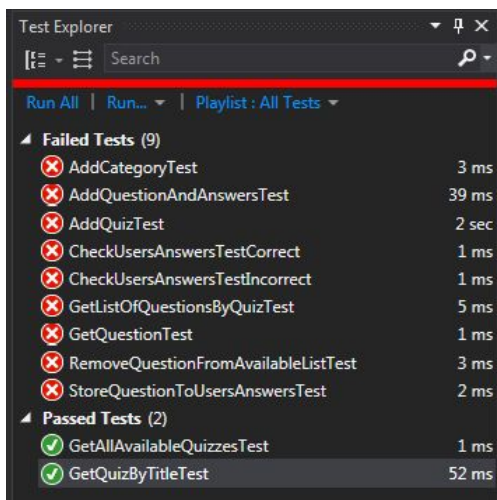
```
8 references | AoifePearson@DSTOUPUT.co.uk, 19 hours ago | 1 author, 1 change
public class QuizViewModel
{
    private QuizManagerEntities QuizEntities = new QuizManagerEntities();

    2 references | AoifePearson@DSTOUPUT.co.uk, 19 hours ago | 1 author, 1 change
    public QuizViewModel()
    {
    }

    2 references | 1/2 passing | AoifePearson@DSTOUPUT.co.uk, 19 hours ago | 1 author, 1 change
    public List<Quiz> GetAllAvailableQuizzes()
    {
        return QuizEntities.Quizzes.ToList();
    }

    1 reference | 1/1 passing | AoifePearson@DSTOUPUT.co.uk, 19 hours ago | 1 author, 1 change
    public Quiz GetQuizByTitle(string quizTitle)
    {
        return QuizEntities.Quizzes.Where(x => x.Title == quizTitle).Single();
    }
}
```

The GetAllAvailableQuizzes Method will output the Quiz data from the DB toList. And the GetQuizByTitle uses a LINQ statement to find the Quiz from the list of quizzes that matches the searched title.

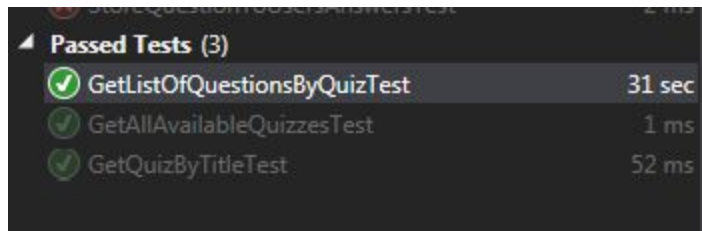


Both tests passed.

TakeQuizViewModel

I then started to write the code necessary to pass the `GetListOfQuestionsByQuizTest`, to do this I first created a private `quizViewModel` in the current class so that I could get the current quiz. I then used the existing Foreign Key to reference the Quizzes Questions to a list of Question Objects.

I used this method to initialise the `AvailableQuestionsByNumber` dictionary so that this method can be used to get the numbered list of questions for use when generating questions.



Passed tests.

The next method I created was the `GetRandomQuestion()`. It occurred to me that I could not generate a random number between `x` and `y` because it would not skip numbers. If I had a list of available numbers 1,2,3,4,5 and a 3 was pulled out on the first try, then the syntax `random(1, list.length)` would produce numbers 1,2,3,4 instead of 1,2,4,5 that I was expecting.

Instead, I decided to use the keys of the `AvailableQuestionsByNumber` as an array of integers, then create a random number for the position in the array rather than the value itself. This way it would always pull out a value of a question that had not already been taken.

```

2 references | 1/1 passing | 0 changes | 0 authors, 0 changes
public Question GetRandomQuestion()
{
    Random random = new Random();
    int randomNumber = 0;
    if (AvailableQuestionsByNumber.Count > 1)
    {
        List<int> listOfAvailableKeys = new List<int>();
        foreach (var item in AvailableQuestionsByNumber)
        {
            listOfAvailableKeys.Add(item.Key);
        }
        int[] arrayOfAvailableKeys = listOfAvailableKeys.ToArray();
        do
        {
            randomNumber = arrayOfAvailableKeys[random.Next(1, arrayOfAvailableKeys.Length)];
        } while (AvailableQuestionsByNumber.Any(x => x.Key == randomNumber) == false);
    }
    else
    {
        randomNumber = AvailableQuestionsByNumber.Single().Key;
    }

    var question = AvailableQuestionsByNumber.Where(x => x.Key == randomNumber).Single();
    return question.Value;
}

```

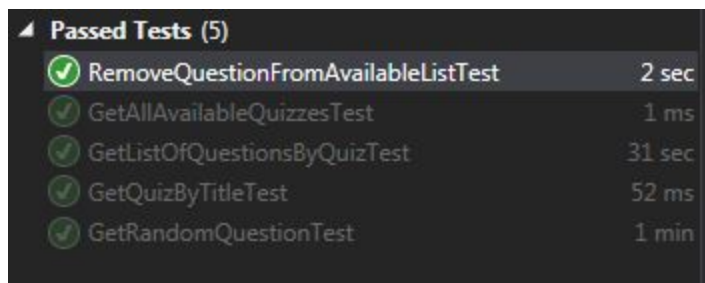
Once I had the position of the number in the list, I could use LINQ to match the key to the random number and return the related question. To save on runtime I added the if, else so that if there was only 1 question left then that question would be automatically selected.

Run All Run... ▾ Playlist: All Tests ▾		
▲ Failed Tests (7)		
✗ AddCategoryTest		3 ms
✗ AddQuestionAndAnswersTest		39 ms
✗ AddQuizTest		2 sec
✗ CheckUsersAnswersTestCorrect		1 ms
✗ CheckUsersAnswersTestIncorrect		1 ms
✗ RemoveQuestionFromAvailableListTest		3 ms
✗ StoreQuestionToUsersAnswersTest		2 ms
▲ Passed Tests (4)		
✓ GetRandomQuestionTest		1 min
✓ GetAllAvailableQuizzesTest		1 ms
✓ GetListOfQuestionsByQuizTest		31 sec
✓ GetQuizByTitleTest		52 ms

This test passed, however, it does not pass every time. Since the AvailableQuestionsByNumber is not being updated sometimes the same number will be pulled out twice. This is where the RemoveQuestionFromAvailableList() comes in.

```
1 reference | 1/1 passing | AoifePearson@DSTOUPUT.co.uk, 20 hours ago | 1 author, 1 change
public void RemoveQuestionFromAvailableList(Question question)
{
    var dictionaryQuestion = AvailableQuestionsByNumber.Where(x => x.Value == question).Single();
    AvailableQuestionsByNumber.Remove(dictionaryQuestion.Key);
}
```

For this method, I simply created a local var to store the question from the AvailableQuestionsByNumber, found using LINQ statement, and then remove the item from the list.



Passed Tests (5)	
RemoveQuestionFromAvailableListTest	2 sec
GetAllAvailableQuizzesTest	1 ms
GetListOfQuestionsByQuizTest	31 sec
GetQuizByTitleTest	52 ms
GetRandomQuestionTest	1 min

All tests passed.

I started to code the final 2 methods in the TakeQuizViewModel. StoreQuestionToUsersAnswers and CheckUsersAnswers.

For the StoreQuestionToUsersAnswers () I had to add a local property to the Answer class, a bool named IsSelected, which I will amend to true/false when the user selects the checkbox on the Quiz interface.

I had to create a new property CurrentAnswers, to store the current answers to the question that has been selected. This is because the answers selected on the front end and have not interacted with any stored variables currently, therefore the questions foreign key answers will not know which was selected by the user. Storing the answers this way allows me to alter the IsSelected property and reference it to check it's value.

Once I have found the local version of the answer that has been selected, I can then use the Id of that answer to find the DB version. I can then add the question and the DB data version of the answer(since this will not change, whereas the CurrentAnswers property will change with every new question that is answered) to the UsersAnswers dictionary.

```

3 references | 1/3 passing | AoifePearson@DSTOUTPUT.co.uk, 20 hours ago | 1 author, 1 change
public void StoreQuestionToUsersAnswers(Question question)
{
    Answer selectedAnswer = CurrentAnswers.Where(x => x.IsSelected == true).Single();
    Answer dbSelectedAnswer = QuizEntities.Answers.Where(x => x.Id == selectedAnswer.Id).Single();
    UsersAnswers.Add(question, dbSelectedAnswer);
}

```

Passed Tests (6)	
✓ StoreQuestionToUsersAnswersTest	2 sec
✓ GetAllAvailableQuizzesTest	1 ms
✓ GetListOfQuestionsByQuizTest	31 sec
✓ GetQuizByTitleTest	52 ms
✓ GetRandomQuestionTest	1 min
✓ RemoveQuestionFromAvailableListTest	2 sec

The tests all passed.

Finally, I created the CheckUsersAnswers () functionality.

```

2 references | 2/2 passing | AoifePearson@DSTOUTPUT.co.uk, 20 hours ago | 1 author, 1 change
public void CheckUsersAnswers(Dictionary<Question, Answer> usersAnswers)
{
    List<Answer> CorrectAnswers = new List<Answer>();
    List<Answer> IncorrectAnswers = new List<Answer>();
    foreach (var answer in usersAnswers)
    {
        if (answer.ValueCorrect == 1)
        {
            CorrectAnswers.Add(answer.Value);
        }
        else
        {
            IncorrectAnswers.Add(answer.Value);
        }
    }
    int totalQuestions = IncorrectAnswers.Count + CorrectAnswers.Count;
    int passPercentage = (CorrectAnswers.Count / totalQuestions) * 100;

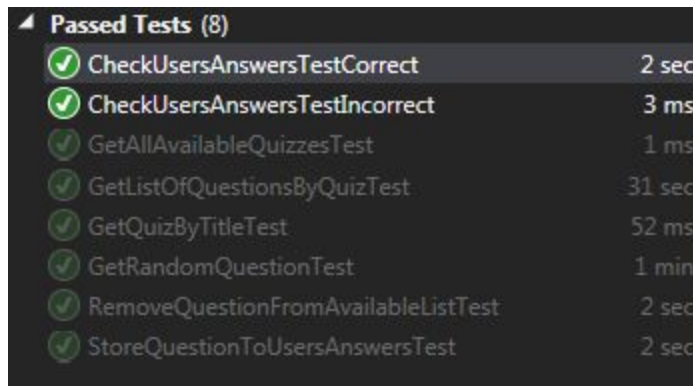
    if (CorrectAnswers.Count >= CurrentQuiz.PassMark)
    {
        FinalScore = string.Format("Congratulations You passed! \n You Scored {0}%", passPercentage);
    }
    else
    {
        FinalScore = string.Format("We're sorry, you did not pass. \n You Scored {0}%", passPercentage);
    }
}

```

This method will separate the UsersAnswers into 2 lists, one where the correct property is set to 1, and the other to 0. If the answer is 1 then the answer was correct, if 0 then it was incorrect. It

will then find the total questions and the percentage pass rate for the quiz, ready to be printed in the FinalScore.

The code will then find if the count of the correct answers is greater or less than the PassMark for the quiz. If greater it will return the FinalScore as the “Congratulations” message along with the pass percentage rate. If less then it will return the FinalScore as the “we’re sorry” message along with the pass percentage rate.



Passed Tests (8)	
✓ CheckUsersAnswersTestCorrect	2 sec
✓ CheckUsersAnswersTestIncorrect	3 ms
✓ GetAllAvailableQuizzesTest	1 ms
✓ GetListOfQuestionsByQuizTest	31 sec
✓ GetQuizByTitleTest	52 ms
✓ GetRandomQuestionTest	1 min
✓ RemoveQuestionFromAvailableListTest	2 sec
✓ StoreQuestionToUsersAnswersTest	2 sec

All tests passed.

Quiz Admin

To create the Quiz Admin methods I started with the Add Quiz functionality. This method would need to check that the Quiz being added did not already exist to ensure no duplicates would be generated when the test was run. Once it was established that the quiz did not already exist, the method sets the different values of the quiz appropriately, adds the new quiz to the EDMX DB QuizEntities and saves the changes. This ensures that changes made at this point are reflected in the DB on my local machine.

```
1 reference | 1/1 passing | AoifePearson@DSTOUTPUT.co.uk, 21 hours ago | 1 author, 1 change
public void AddQuiz(Quiz newQuizItem)
{
    bool alreadyExists = QuizEntities.Quizzes.Any(x => x.Title == newQuizItem.Title);
    if (alreadyExists == false)
    {
        var listOfAllQuizzes = quizViewModel.GetAllAvailableQuizzes();
        var selectedCategory = QuizEntities.Categories.Where(m => m.Title == newQuizItem.Category.Title).Single();
        newQuizItem.Category = selectedCategory;
        newQuizItem.CategoryId = selectedCategory.Id;
        newQuizItem.Id = listOfAllQuizzes.Last().Id + 1;
        newQuizItem.DateCreated = DateTime.Now;
        QuizEntities.Quizzes.Add(newQuizItem);
        QuizEntities.SaveChanges();
    }
}
```

Passed Tests (9)	
✓ AddQuizTest	7 sec
✓ CheckUsersAnswersTestCorrect	2 sec
✓ CheckUsersAnswersTestIncorrect	3 ms
✓ GetAllAvailableQuizzesTest	1 ms
✓ GetListOfQuestionsByQuizTest	31 sec
✓ GetQuizByTitleTest	52 ms
✓ GetRandomQuestionTest	1 min
✓ RemoveQuestionFromAvailableListTest	2 sec
✓ StoreQuestionToUsersAnswersTest	2 sec

Next, I built the Add Categories() which would set the string of “category Name” as the title property of the category object, and the method would assign appropriate values for the other properties. The new category item would then be added to the EDMX DB, then saved to the LocalMachine DB.

```
1 reference | 1/1 passing | AoifePearson@DSTOUTPUT.co.uk, 21 hours ago | 1 author, 1 change
public void AddCategory(string categoryName)
{
    List<Category> currentCategories = QuizEntities.Categories.ToList();
    Category newCategory = new Category();
    newCategory.Title = categoryName;
    newCategory.Id = currentCategories.Last().Id + 1;
    QuizEntities.Categories.Add(newCategory);
    QuizEntities.SaveChanges();
}
```

Passed Tests (10)	
✓ AddCategoryTest	2 sec
✓ AddQuizTest	7 sec
✓ CheckUsersAnswersTestCorrect	2 sec
✓ CheckUsersAnswersTestIncorrect	3 ms
✓ GetAllAvailableQuizzesTest	1 ms
✓ GetListOfQuestionsByQuizTest	31 sec
✓ GetQuizByTitleTest	52 ms
✓ GetRandomQuestionTest	1 min
✓ RemoveQuestionFromAvailableListTest	2 sec
✓ StoreQuestionToUsersAnswersTest	2 sec

Finally, I created the AddQuestionsAndAnswers(). Here the Question would first go through set up as a new Question Object and its properties would be set appropriately, then the changes saved to the DB. Once the question was Added then the Answers would be added the same

way. I had to do this in a specific order since the Answers rely on a Foreign Key QuestionId, which would only be present if the question was already added to the DB.

```
1 reference | 0/1 passing | AoifePearson@DSTOUPUT.co.uk, 21 hours ago | 1 author, 1 change
public void AddQuestionAndAnswers(Question question, List<Answer> answers)
{
    List<Question> currentListOfQuestions = QuizEntities.Questions.ToList();
    question.Id = currentListOfQuestions.Last().Id + 1;
    question.QuizId = QuizEntities.Quizzes.ToList().Last().Id;
    question.Quiz = QuizEntities.Quizzes.Where(m => m.Id == question.QuizId).Single();
    QuizEntities.Questions.Add(question);
    QuizEntities.SaveChanges();

    foreach (var answer in answers)
    {
        List<Answer> currentListOfAnswers = QuizEntities.Answers.ToList();
        answer.QuestionId = question.Id;
        answer.Id = currentListOfAnswers.Last().Id + 1;
        if (answer.IsCorrect == true)
        {
            answer.Correct = 1;
        }
        else
        {
            answer.Correct = 0;
        }

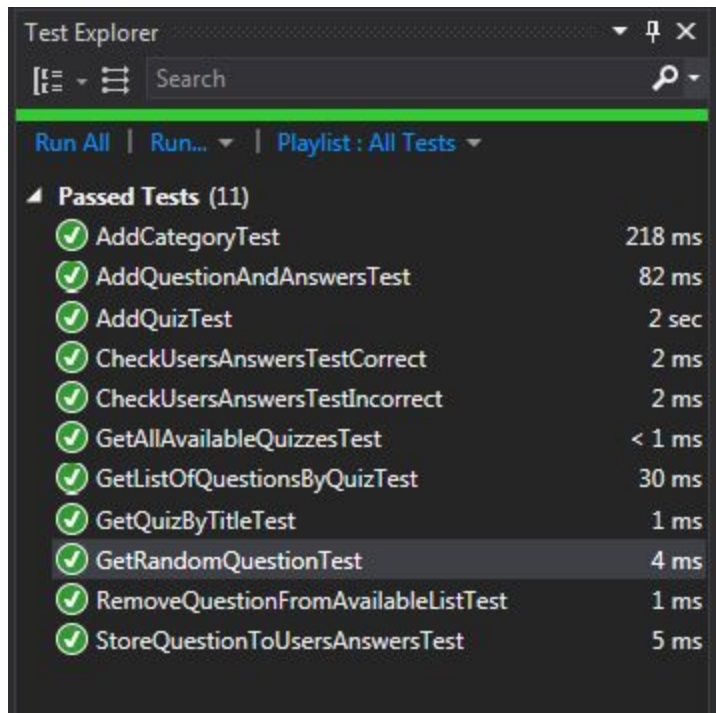
        QuizEntities.Answers.Add(answer);
        QuizEntities.SaveChanges();
    }
}
```

Passed Tests (11)

✓ AddQuestionAndAnswersTest	2 sec
✓ AddCategoryTest	2 sec
✓ AddQuizTest	7 sec
✓ CheckUsersAnswersTestCorrect	2 sec
✓ CheckUsersAnswersTestIncorrect	3 ms
✓ GetAllAvailableQuizzesTest	1 ms
✓ GetListOfQuestionsByQuizTest	31 sec
✓ GetQuizByTitleTest	52 ms
✓ GetRandomQuestionTest	1 min
✓ RemoveQuestionFromAvailableListTest	2 sec
✓ StoreQuestionToUsersAnswersTest	2 sec

The test had passed.

I ran all the Tests at this point:



And all Tests Passed.

Views

Home Page

Now that the functional code is present, I can start building my views. I began with the Home Page:

Quiz Manager

Home

Add Quizes

Hello EditUser@test.com!

Log off

All Quizes

Here you can search for and view all available quizzes

Search for Quiz by Title

Search

Title	Author	Description
Test Quiz	By Aoife	This is a Test Quiz
TESTQUIZ2	By Aoife	description
New Test Quiz	By Aoife	New Test Description

© 2019 - Quiz Manager Application

Here I built the darker themed Navbar and added the Quiz Manager text as a placeholder for a Navbar Brand. If the site were to adopt a logo at a later date then this can be added here.

I followed my wireframe designs and used the inbuilt MVC Razor tools to build elements such as navigation links and text boxes.

I added the navbar and footer to a Layout Page, so these elements will be displayed on every page except for the Login screen which I have set the layout to equal null.

Quiz Info

[Quiz Manager](#) [Home](#) [Add Quizes](#) Hello EditUser@test.com! [Log off](#)

Test Quiz

This is a Test Quiz

Pass Mark: 1

Author: Aoife

[Start Quiz](#)

© 2019 - Quiz Manager Application

This is the page the user will see once they click on the Title for a Quiz from the Home Page.

Following the details provided in the Wireframe, I was able to create this view.

Question Page

[Quiz Manager](#) [Home](#) [Add Quizes](#) Hello EditUser@test.com! [Log off](#)

Is this still a question?

A.No ☐

B.No ☐

C.No ☐

D.Yes ☐

[Next](#)

© 2019 - Quiz Manager Application

I added the functionality for the individual answers by creating a 'for loop' of the `Model.CurrentAnswers` and then referencing the `i` attribute, either as `if i== 0` {<p>A. </P>} or as `m.CurrentAnswers[i].Answer1`. Each time the user clicks the next button it will call the `GetRandomQuestion()` and return the questions in a random order every time.

[Quiz Manager](#) [Home](#) [Add Quizes](#) Hello EditUser@test.com! [Log off](#)

Is This a Test Question?

A.No

☐

B.No

☐

C.No

☐

D.Yes

☐

[Finish Quiz](#)

By adding an if into the cshtml that checks whether the `AvailableQuestionsByNumber` count is equal to 1 I was able to change the button for the final question screen.

Finish Quiz

[Quiz Manager](#) [Home](#) [Add Quizes](#) Hello EditUser@test.com! [Log off](#)

QuizFinish

Congratulations You passed! You Scored 100%

Question	Answer	Result	Explanation
Is this still a question?	Yes	Correct	This is Correct: The Question is a Test Question
Is this still a question?	Yes	Correct	This is Correct: The Question is a Test Question
Is This a Test Question?	Yes	Correct	This is Correct: The Question is a Test Question

Home

© 2019 - Quiz Manager Application

Here I have built the Finish Quiz View. In the above screenshot, I have logged in as an Edit User. As a result, I can see the Q&A pairs, the result and the explanation for each question on the quiz.

Restricted User View

[Quiz Manager](#) [Home](#) Hello RestrictedUser@test.com! [Log off](#)

All Quizes

Here you can search for and view all available quizzes

Search for Quiz by Title[Search](#)

Title	Author	Description
Test Quiz	By Aoife	This is a Test Quiz
TESTQUIZ2	By Aoife	description
New Test Quiz	By Aoife	New Test Description

© 2019 - Quiz Manager Application

By making use of the MVC Roles Attribute, I was able to add an if statement to the Layout page to ensure that Only Edit Users could see the Add Quiz tab at the top of the screen.

```

<li class="nav-item">@Html.ActionLink("Home", "Index", "Home")</li>
@if (User.IsInRole("Edit"))
{
    <li class="nav-item">@Html.ActionLink("Add Quizes", "AddQuiz", "Home")</li>
}
/ul>
@Html.Partial("_LoginPartial")
>

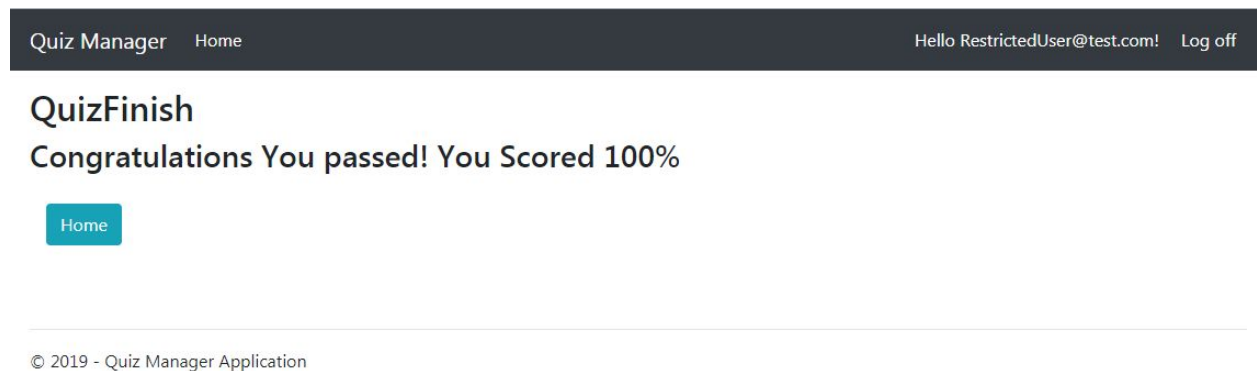
```

I added a similar if to the Quiz Finish Page allowing only View and Edit Users the ability to see the Q&A Pairs table:

```

<h2>QuizFinish</h2>
<h3>@Model.FinalScore</h3>
@if (User.IsInRole("View") || User.IsInRole("Edit"))
{
    <table class="table">
        <thead>

```



The View Users will only see their Pass/Fail score now.

As a final level of authorization, I added the Authorize Roles attribute to the Home page. This will prevent any users who have not logged on from accessing the site.

```

0 references | AoifePearson@DSTOUPUT.co.uk, 1 hour ago | 1 author, 2 changes
public class HomeController : Controller
{
    [Authorize(Roles = "Restricted, Edit , View")]
    0 references | AoifePearson@DSTOUPUT.co.uk, 1 hour ago | 1 author, 1 cha
    public ActionResult Index(QuizViewModel model)
    {
        if (model.SearchByQuizTitle == null)

```

Please see Video QuizManagerDay2.mp4 to see my progress up to this point

3rd July

Today I began working on the Quiz Edit User views. I decided to build this as the process would flow in a real run of the system to ensure I did not forget to create any views.

QuizAdmin Controller

Add Quiz

The first view I built was the add Quiz view

[Quiz Manager](#) [Home](#) [Add Quizes](#) Hello EditUser@test.com! [Log off](#)

Add Category

Title

Add Quiz

Title

Description

Author

PassMark

Category

© 2019 - Quiz Manager Application

Using MVC forms I was able to create 2 different forms on the same view, 1 for the Add Category and one for the Add Quiz. The Add Category form will redirect to the ActionResult “AddCategory”, here the QuizAdmin.cs method Add Category() will be called to add a new category to the DB.

This will then redirect back to the “AddQuiz” Action Result, the AddCategory does not need its own view since the functionality is self-contained and the AddQuiz form still needs to be visible once a new category has been added.

I used a new property in the QuizViewModel so that the form could use the TextBoxFor HTMLHelper to assign a value to the new category item.

The Add Quiz Form required a few more steps. First I had to create a list of categories the user could choose from using a dropdown. To do this I created a new List<Category> and using the QuizViewModel constructor, populated it with the categories from the DB. this way, every time the QuizViewModel would be called, it would update the list of categories.

Using the HTML helper DropDownFor allowed me to create a dropdown for the available categories that the user could choose from.

All the values for the textboxes and dropdowns assigned the values to a new property I created “NewQuizItem”, this new item is what I used to call the QuizAdmin AddQuiz().

Add Questions

Quiz Manager

Home

Add Quizzes

Hello EditUser@test.com! Log off

Add Questions

Question

Answer		Explanation		Correct	<input type="checkbox"/>
Answer		Explanation		Correct	<input type="checkbox"/>
Answer		Explanation		Correct	<input type="checkbox"/>
Answer		Explanation		Correct	<input type="checkbox"/>
Answer		Explanation		Correct	<input type="checkbox"/>

Add Question

Finish Creating Quiz

© 2019 - Quiz Manager Application

Following the same process as before when creating MVC forms, I added 2 new properties to the QuizViewModel. Questions and ArrayOfAnswers, the Question property would be populated using the textbox for helper on the Question textbox, and the array of answers would each be populated by the text boxes below.

I wanted to ensure that the users could add up to 5 questions as per the spec, but if some textboxes were left empty I did not want those to be added to the DB. So I created a foreach loop in the AddQuestionAndAnswers() which would create a new cleanAnswerList containing only the answers with a value.

I then ran this new list through the add to DB code.

```
// cleanup list
List<Answer> cleanAnswerList = new List<Answer>();
foreach (var answer in answers)
{
    if (answer.Answer1 != null)
    {
        cleanAnswerList.Add(answer);
    }
}
// run clean list
foreach (var answer in cleanAnswerList)
{
```

When the User clicks Finish Quiz, they are returned to the Home Page and the new quiz is in the list.

Edit Quiz

The first step in creating the Edit Quiz functionality was to add a link to the edit page that only edit users could see.

[Quiz Manager](#) [Home](#) [Add Quizes](#) Hello EditUser@test.com! [Log off](#)

All Quizes

Here you can search for and view all available quizes

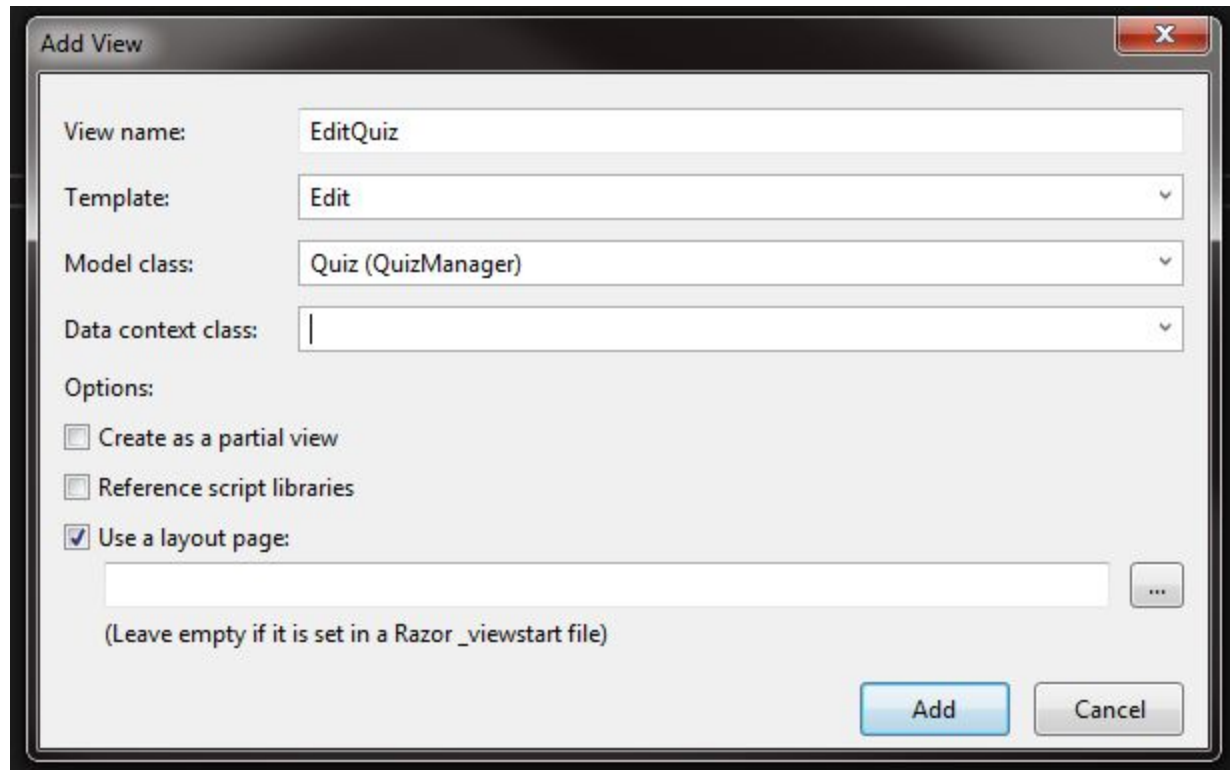
Search for Quiz by Title

Title	Author	Description	
Test Quiz	By Aoife	This is a Test Quiz	Edit Quiz
TESTQUIZ2	By Aoife	description	Edit Quiz

This would be used to redirect to the ActionView "EditQuiz".

To Create the View for the Edit QUIz, I set up a new view with the Template set to Edit, and the Quiz class selected as the model.

This way the Quiz data would be saved if fields were left un-edited and the edited fields would be updated by the model.



This generated a form for me that would allow me to edit any field related to the Quiz class. I changed the fields that I did not want to edit to HTML.hidden for so that they would retain their original values. I then created a new property in the Local Quiz class called Categories, which was a list of Category.

This allowed me to create a drop down list which I populated using the QuizViewModel Categories. I could not use the existing list as it was part of a different view. Instead, I assigned the list values to this new list.

I created a table to display the Q&As for the quiz (editing these would come later) and made the form return to the EditQuiz Action Result to check that the values would update on submit. Currently, they are not saved anywhere since I had not created the save to DB functionality.

Edit Quiz

Quiz Details

Title	Description	PassMark	Category	
<input type="text" value="Test Quiz"/>	<input type="text" value="This is a Test Quiz"/>	<input type="text" value="1"/>	<input type="text" value="TESTCategory"/>	<input type="button" value="Save"/>

Question Details

Question	Answer	Explanation	Correct
Is This a Test Question?	No	This is Incorrect: The Question is a Test Question	0
	No	This is Incorrect: The Question is a Test Question	0
	No	This is Incorrect: The Question is a Test Question	0
	Yes	This is Correct: The Question is a Test Question	1
Is this still a question?	No	This is Incorrect: The Question is a Test Question	0
	No	This is Incorrect: The Question is a Test Question	0
	No	This is Incorrect: The Question is a Test Question	0
	Yes	This is Correct: The Question is a Test Question	1
Is this still a question?	No	This is Incorrect: The Question is a Test Question	0
	No	This is Incorrect: The Question is a Test Question	0
	No	This is Incorrect: The Question is a Test Question	0

Using the Quiz class as a View Model was working up to the point where I needed to edit the Questions. I was unable to retain the Questions and Answer values using that class. So to fix the problem I decided to create a new ViewModel “EditQuizViewModel”, which would be used by this section of the system. During my first attempt, I was trying to avoid repeating code and properties that were already available, however, It became more apparent that I would need a completely new View model to work with as I was required to retrieve information, amend the fields and then save the changes. All three sections needed to happen within a contained system to avoid ‘Code Spaghetti’ as I have dubbed it.

Now that I had an understanding of how the System should work together, I created my unit tests to test the new methods I would be creating. For the Edit Quiz functionality, I ended up coding a SetQuizVariables() and a SaveQuizDetails(). So, I would do the same for the EditQuestions.

After writing my unit tests for:

- SetQuizVariablesTest()
- SetQuestionVariablesTest()

- SaveQuizDetailsTest()
- SaveQuestionDetailsTest()

I ensured all tests passed before moving on with development.

Quiz Manager

Home

Add Quizzes

Hello EditUser@test.com!

Log off

Edit Quiz

Quiz Details

Title

Description

PassMark

Category

Save Quiz Details

Title

Test Quiz

Description

This is a Test Quiz

PassMark

3

Category

TESTCategory

Question Details

Question	Answer	Explanation	Correct	
Is This a Test Question?	No	This is Incorrect: The Question is a Test Question	Incorrect	Edit
	No	This is Incorrect: The Question is a Test Question	Incorrect	
	No	This is Incorrect: The Question is a Test Question	Incorrect	
	Yes	This is Correct: The Question is a Test Question	Correct	
Is this still a question?	No	This is Incorrect: The Question is a Test Question	Incorrect	Edit
	No	This is Incorrect: The Question is a Test Question	Incorrect	
	No	This is Incorrect: The Question is a Test Question	Incorrect	
	Yes	This is Correct: The Question is a Test Question	Correct	
Is this still a question?	No	This is Incorrect: The Question is a Test Question	Incorrect	Edit
	No	This is Incorrect: The Question is a Test Question	Incorrect	

I also decided to clean up my views and add an edit button to the Questions Table. This would mean that the individual questions could be edited without the noise of the other fields on the screen, Improving the UI.

Quiz ManagerHomeAdd Quizes

Hello EditUser@test.com!Log off

Edit Question

Question

Is This a Test Question?

Answer	No	Explanation	This is Incorrect: The Question is a Ti	Correct	<input type="checkbox"/>
Answer	No	Explanation	This is Incorrect: The Question is a Ti	Correct	<input type="checkbox"/>
Answer	No	Explanation	This is Incorrect: The Question is a Ti	Correct	<input type="checkbox"/>
Answer	Yes	Explanation	This is Correct: The Question is a Tes	Correct	<input checked="" type="checkbox"/>

Save

© 2019 - Quiz Manager Application

Now the User can select a Quiz and edit the details, then select any Question to Edit. Once they have made their changes to the Question they are returned to the EditQuizPage and the changes are reflected in the Questions Table. Changes saved to DB.

4th July

Testing

Now that I had coded my project, I could run my manual tests against what I had created. (please see Manual Test cases PDF in deliverables folder)

To run a manual test, you follow the steps on the left-hand side of the table, and if the expected result is what happens when you type Passed into the Pass/Fail field. If something unexpected happens, you type Fail and provide an explanation of your findings. These will become bugs later on.

Bugs

One bug that I found was that the percentage pass rate was not being calculated correctly for the user. The correct message would always display and the percentage would be either a 0% or 100%.

I put a breakpoint in before the CheckUsersAnswers() was called and debugged the application.

```

let totalQuestions = IncorrectAnswers.Count + CorrectAnswers.Count;
let passPercentage = (CorrectAnswers.Count / totalQuestions) * 100;
return passPercentage;

```

I discovered that my percentage was not being set as a proper decimal. Which is when I realised I had set the pass percentage to an int. This would prevent the decimal from ever becoming usable since all numbers would be rounded before calculation. By changing my ints to decimals I was able to retrieve the correct values, and then round the number to 0 decimal places to receive a user-friendly number.

[Quiz Manager](#)
[Home](#)
[Add Quizes](#)
Hello EditUser@test.com! [Log off](#)

QuizFinish

We're sorry, you did not pass. You Scored 67%

Question	Answer	Result	Explanation
Is This Still a question?	Yes	Correct	This is Correct: The Question is a Test Question
Is this still a question?	No	Incorrect	This is Incorrect: The Question is a Test Question
Is this a Test Question?	Yes	Correct	This is Correct: The Question is a Test Question

Home

© 2019 - Quiz Manager Application

5th July

I wrote the User guide today and arranged my documentation up to this point ready to hand in.