Retrospective Document

WebbiSkools Quiz Manager

Aimee Craig

2019

Contents

[Overview 2](#_Toc24103890)

[Offline Use 2](#_Toc24103891)

[Functional And Unit Tests 2](#_Toc24103892)

[Test Database 2](#_Toc24103893)

[Limitations 2](#_Toc24103894)

[Reordering Questions 2](#_Toc24103895)

[Delete Question/Answer Confirmation 2](#_Toc24103896)

[Front End Validation 2](#_Toc24103897)

[Future Improvements 2](#_Toc24103898)

[Drag And Drop 3](#_Toc24103899)

[Front End Validation 3](#_Toc24103900)

[Functional Tests Database Teardown 3](#_Toc24103901)

# Overview

The project was completed within the deadline and all requested functionality was implemented. A demo of the functionality can be seen by viewing the “Demo.mp4” video in the “Documentation” folder. Below are features that I am particularly proud of.

## Offline Use

This project is built in such a way that it can be used completely offline. The database containing the quizzes, questions, answers and users is created and run from the user’s machine. This does mean that quizzes created on one machine won’t be accessible to someone running the application on another machine but this can be resolved by deploying the application if need be.

## Functional And Unit Tests

I was able to include functional and unit tests within the project by creating these as I developed the product. This meant that my test coverage stayed high throughout the project and I was able to implement new features quicker as I could be confident that my changes had not inadvertently broken anything.

## Test Database

My application has two environments from which to run from, a development environment and a production environment. This means that when the user is running the application in the development environment, the application is connecting to a different database altogether. Because of this the user can confidently make changes to the database without touching live data.

This also means that my functional tests can run against a test database. I can be sure of the expected outcome of my functional tests as I know that users won’t be able to add or delete quizzes in the database I am testing against.

# Limitations

## Reordering Questions

Currently, the user is unable to reorder questions in either the create or edit pages. The user can add a question at any point in the sequence of questions but say for instance the user wants to move the second question to the end of the list of questions, they would have to delete all the questions after the second question and then re-add them after the first question.

## Delete Question/Answer Confirmation

When a user clicks on the Delete Question or Delete Answer button, the question/answer is deleted immediately. The user is not given any chance to confirm the deletion so this could lead to the user accidentally deleting a question/answer.

## Front End Validation

The user is able to create a quiz without a quiz title, questions or answers as there is no front end validation present. This means that blank quizzes could be added to the database and then subsequently be available in the list of quizzes.

I have detailed the issue and possible resolution further in the “Front End Validation” section below.

# Future Improvements

## Drag And Drop

I would like the functionality to drag and drop questions and answers within the create and edit pages as I feel this would improve the experience for the end-user. This would also resolve the limitation on reordering questions.

Fortunately, the functionality to drag and drop elements already exists within the jQuery UI framework which is already present within my solution. To add this functionality I would need to call “.sortable()” and “.disableSelection()” on the div containing the elements I would like to be able to drag and drop (in my case, this would be the div with an Id of “all-questions-and-answers”). More information on jQuery UI Sortable can be found [here](https://jqueryui.com/sortable/).

## Front End Validation

.NET Core 3 MVC already includes the functionality to be able to add front end validation to forms with its jQuery Unobtrusive Validation framework. Unfortunately this gets trickier the more complex the model you are trying to validate. Each input you want to add front end validation for requires a “asp-for” tag helper or a “data-val” attribute (as well as a “data-val-required” or similar attribute depending on what validation you are adding). A span is then required to hold the validation error message and this requires a “asp-validation-for” tag helper or “data-valmsg-for” and “data-valmsg-replace” attributes. The “data-valmsg-for” attribute value needs to match the name attribute of the input to be validated in order to correct validate and display any error messages.

The issue is that my inputs on the page can be dynamically generated and as such I cannot use the “asp-for” tag helper. This is because the “asp-for” tag helper gets changed to name and Id attributes on page render which only occurs once. I would have to instead add the “data-val”, “data-val-required” (if used), “name”, and “id” attributes dynamically as well as the “data-valmsg-for” and “data-valmsg-replace” attributes on the input’s respective span.

However, I could still take advantage of the tag helpers by creating a partial view that contains a new set of question and answers inputs that use the “asp-for” tag helper and then add this to the page via an AJAX request. This would ensure that all attributes required for the front end validation framework to work would be included on the page.

## Test Database TearDown

Although my functional tests are running against a separate database, there is currently no way for me to clear the database before each test. This means that I cannot be sure of the number of quizzes in the database to make assertions against. To combat this, I have added manual tests to my test plan that will check that when a quiz is created/edited/deleted that it is displayed appropriately in the application.

As the database is set up on application start and the application has to be started before the tests run, I cannot replace the database with an in memory database as I have done for the unit tests.

## Registration

Although registration wasn’t a requirement of this project, I feel as though including a registration system of some kind would be useful in the future. Currently, new users have to be added to the database initialisation method and they will only be added if the database is empty at the time of running.

By adding a registration system, users could be added to the database without having to amend the DbInitialiser.cs file and then running commands in the command line interface to delete the database first. This also means that users will be able to keep their current list of quizzes when adding a new user.