Iteration: 6 (30/03 - 05/04)

Alex Taylor - amt22@aber.ac.uk

April 5, 2017

Version 0.4 (Draft)

Contents

1 Story: The Admin can show the results of the question in a sensible format e.g. graph

1.1 Analysis - Breakdown of Tasks

- Save the data
- Ensure no users submit twice
- Read from that data in the admin panel
- Prettify this results tab

1.2 Design

TODO: add design images here of the admin panel

1.3 Implementation

There was quite a lot of work to do in this story, the first item in particular was the majority of work. At first it was decided that a simple database could be used to store all the responses however it was soon realised that users could probably submit more than once if we dont associate users with a response. The best way to deal with users that dont login is to use the laravel session cookie to identify them anonymously.

To store all these responses in a database would be quite a lot of data, rather than having say six rows for a question with six answers that just tallies up the answers the actual session name had to be stored with the answer given so it could be changed. This meant that now there would be a row per answer, equalling at worst case 300 rows per question. This was thought to be a bit drastic so an alternative was suggested: to use a CSV file with each row having the user and an answer, with each question being a seperate csv file. These would be stored locally under a session folder in the public directory. These files would be created and destroyed during quiz cycle. This method would also be a good for the downloading of answers from another story.

Ultimately however, trying to write all this data to a csv was too troublesome to be worth it. If a user changed their answer then the entire csv had to be copied with one line changed, editing a single line is not possible with PHP. Therefore it was decided that going back to a database based method would be better, with the idea that at the end of a quiz, all the data would be deleted.

An answers table was created that specified the session, question, user and the answer given. The first three of those were used in a unique composite key to prevent multiple answers from the same users on the same questions in the same session. The model was then written to save this data when data was POSTed to the /results url. Other functions include those to delete the data when a quiz was ended and various association functions.

To read this data on the front end, an AJAX request was set up from an admin panel button that GETs the same results url which calls an action to render the results data in a JSON format. The results are in a very basic format in a key, value pairs of answer, total number of selections. TODO: maybe talk about this more

This JSON data could then be used in the ChartJS library which was selected to render the results bar chart on the page.

1.4 Testing

2 Story: Users should not be able to submit their own answers by altering the HTML, as they did with Qwizom

2.1 Analysis - Breakdown of Tasks

Very few tasks as will be explained in the design section:

- Add check for empty arrays on frontend
- Add check for empty arrays on backend

2.2 Implementation

Thanks to the design of the previous section, changing the data meant an empty key was sent if the user tried to change the data submitetd to the server. This could be checked in either the JavaScript or PHP end, but it was decided both would be the best for maximum security.

2.3 Testing

3 Story: The Admin can see what percentage of users connected to the session have answered

3.1 Analysis - Breakdown of Tasks

- Get total number of responses
- Post this number on the page as part of the results form

3.2 Design

This should appear above the results table, possibly as part of the title.

3.3 Implementation

After attempting to get a number of users connected to the channel it became clear that this number is not readily accessible for public channels. It can be gotten via the Pusher API but implementing that would mean locking some functionality down with Pusher and the long term plan is probably going to involve changing to a Redis based WebSocket system. Therefore this number has just been changed to the total number of responses which should give a good idea to the lecturer of what percentage of users have responded. This was simply added to the title attribute of the chart.

3.4 Testing