Iteration: 2 (02/03 - 08/03)

Alex Taylor - amt22@aber.ac.uk

March 18, 2017

Version 0.6 (Draft)

Contents

1 Story: They can create a new quiz in the backend

1.1 Analysis - Breakdown of Tasks

- Need quiz/new form to add name
- Add validation
- This then redirects to quiz.show for this new quiz
- This page needs a button that can add a question

1.2 Design

The design is split into two parts, design for the quiz/new page and a design for the question/new page: (TODO: insert images for these, maybe some form of activity diagram?)

1.3 Implementation

A create page was added for quizzes, this was linked to from the home page. This was mapped to /quizzes/create address. A simple form was added to this page with the name and description of the quiz. Questions would be added on an individual quiz page. The form sends the form as an HTTP POST request to the /quizzes page which then maps onto the 'Store' action in the Quiz Controller. This action is used for server side validation and passing the data to the Quiz model to be saved to the database. The model function called just saves to the database using Eloquent, and returns the id of the new quiz. The user_id is assigned to the quiz by simply getting the current user. Using this id, this Store function redirects to the newly created quiz.

On a quiz page, it displays the name and description and has a button to add any questions to the quiz. This button links to the questions/create page that functions in the same way as the quiz creation, albeit with the relevant fields in the form. The questions are assigned to the quiz by passing the quiz id to the question creation page in a GET variables via the url. This allows the question to be assigned to the quiz during its creation in the database and to be redirected back to the quiz page.

For validation there are two parts, one on the HTML form and the other handles server side. The HTML validation (client side) uses simple HTML 5 validation like the "required" attribute in the form inputs. Further frontend validation could be added utilising Javascript but the HTML solution was far easier to implement, being the addition of one word and not several lines of code that have to be built and added to the correct pages. The server side complements this as client side validation can be circumnavigated by determined users.

The server side validation in Laravel is well supported and made as easy as possible to implement. The Store function used to save data from the POST request simply calls *$this->validate()* on the request data. Inside this validation function, various rules can be imposed on individual pieces of the request data such as simply requiring that is is filled or that it has to have a minimum length (TODO: cite the docs for this).

Something that Laravel enforces is the use of CRSF tokens to prevent cross site request forgery attacks on the site. A token is generated for the project during its creation, and this token must be placed in a hidden field within every form so that it can verify that the authenticated user is the one actually making the requests to the application. (TODO: cite this doc page)

1.4 Testing

Dusk tests:

1. Test to create a new quiz
2. Test the HTML validation for creating a new quiz
3. Test backend validation of quizzes
4. Test a quiz contains questions that have all been added before hand
5. Test to add a question to a quiz
6. Test to make sure the question page contains the relevent information

A problem encountered in this set of tests is the speed of testing. Running these tests takes about a minute. The reason for this is that the tests use DatabaseMigrations rather than DatabaseTransactions, meaning that the database is migrated for every test. Using transactions would reduce the time taken by only doing a migration at the start and then using a transaction for each test. Alternatively having a pre migrated test database on which you run transactions would work too, but this would mean any time you change your database, you would have to reember to migrate the test database.

After attempting to use the transactions it appears as though they are not usable within Dusk. This is because Dusk is running in another process and migrations is the only choice (TODO: cite this with https://github.com/laravel/dusk/issues/110)

2 Story: They can edit an existing quiz they own

2.1 Analysis - Breakdown of Tasks

- There is an edit button on quizzes that links to quiz/edit
- You can edit the name of the quiz
- You can remove questions
- You can add questions
- You can edit the content of a question
- You can delete a quiz and its associated questions (TODO: add this to the trello)

2.2 Design

There was not much design for this stage as it only really adds an edit page which should look the same as the create pages in the above story except that the input fields are pre-filled with data that is being editing.

2.3 Implementation

The edit pages added to the

2.4 Testing

Dusk Tests:

1. Test the deletion of a quiz
2. Test the deletion of a question on a quiz
3. Test the edit page of a quiz
4. Test the edit page of a question

3 Non-Story Work

3.1 Seeding

Some more data needed to be seeded for this section of work, the question data. This involved creating some new Model Factories and using them in the seeders correctly. One issue was trying to create many questions for individual quizzes, but this was overcome using some very basic looping and calling the model factories in the right places.

3.2 Quiz Description

It was decided that quizzes should probably have descriptions attached to them, in case the lecturer needs reminding of what it is in 6 months time. This involved creating a new migration and simply adding a column to the table.