# Iteration: 3 (09/03 - 15/03)

Alex Taylor - amt22@aber.ac.uk

March 18, 2017

Version 0.7 (Draft)

## Contents

**1 Story: Admins can login to the website and run a session with a quiz** **2**
1.1 Analysis - Breakdown of Tasks . . . . . 2
1.2 Design . . . . . 2
1.3 Implementation . . . . . 2
1.4 Testing . . . . . 2

**2 Non-Story Work** **3**
2.1 Refactor Controllers . . . . . 3
2.2 Changing the DB Structure . . . . . 3
2.3 Front-End Setup . . . . . 3

# 1 Story: Admins can login to the website and run a session with a quiz

## 1.1 Analysis - Breakdown of Tasks

This story is quite large and whilst it will probably be split into a number of tasks, the first thing to do was some spike work on the way this system would work. One route is with WebSockets, a relatively new technology that allows the server to push data to the page quickly and easily. This would be nice solution as it is relatively future proof and seems like a better alternative to other solutions that involve a lot of javascript and forcing page changes on the users.

## 1.2 Design

WebSockets were introduced into Laravel 5 and have become one of the defacto ways to update the front end in real time. Unlike in some other web frameworks such as Ruby on Rails the web sockets in Laravel requires some extra set up. In Rails the WebSockets can be run on the main web server that is used to run the site[cite this], in Laravel however another server has to be set up to run these. Laravel offers several different drivers for running the WebSockets, including a Redis server[cite this] and third party application called Pusher. Pusher handles most of the work for you and requires little set up other than creating a free account.

## 1.3 Implementation

Pusher was chosen due to its ease of use and due to its high recommendation rate within the Laravel community. A problem with Pusher is that due to it being a third party service, it is not free forever (it has a number of users limitation). However you could host your own Redis server to mitigate this cost. This means that the system had to be designed in a way that the driver for WebSockets could be changed with ease.

After configuring the spike work application to use Pusher, a simple Laravel Event was created to send a message to the Pusher. To test this event there were a couple methods implemented. The first was to simply register a route that triggers it when the page is visited: event(new App\Events\HelloPusherEvent('Hi there Pusher!')); (put in full line)

Or with a custom made artisan command that can trigger the event. (TODO: insert this command here)

## 1.4 Testing

Both of these methods send the event to Pusher where it is registered: (TODO: insert picture of pusher)

# 2 Non-Story Work

## 2.1 Refactor Controllers

The first major piece of work was to refactor the controllers and models into a far more sensible structure. The problem was that the Eloquent functions for modifying and reading from the database was within the controllers. In a full MVC system, this functionality should be inside the models. To fix this the Eloquent functions were refactored into the respective quiz and question models. This would make it easier if anything ever needed to change within the logic for any database interactions.

## 2.2 Changing the DB Structure

The original design was changed, and the quiz_questions table for linking quizzes and questions together was removed. The original reason for this table was most likely such that questions could be reused. However, after thinking about the potential for that to happen, and the issues that the structure was causing in the model logic it was decided that the quiz_question table was more of a hinderance than a help.

The questions table now simply has a quiz_id column that references the quiz it belongs to. Doing this means that the relationships between the two tables are much easier to define in the models, simply having a belongsTo and hasMany function in both that automatically return the necessary data. Thanks to the previous refactoring of model logic, changing this functionality was quite quick. Deleting rows in the database was also simpler now that there was no quiz_questions table, all the related questions when a quiz is deleted can be deleted at the same time using the onDelete cascade property in the database.

## 2.3 Front-End Setup

This was the first time that any custom css was written and Laravel uses sass to generate its css. To build this sass into css, and also to build any future js Laravel Mix was needed to run builds for this code. For this, npm and node had to be installed so that they could run their webpack build scripts. There were some issues trying to get the build scripts to run, even though it worked on fresh installs of laravel, but eventually a Github issue was discovered that had some solutions. https://github.com/JeffreyWay/laravel-mix/issues/478