Quiz Generator for LASEC Courses

François Farquet

School of Computer and Communication Sciences

Semester Project

June 2013

Responsible
Prof. Serge Vaudenay
EPFL / LASEC

Supervisor
Alexandre Duc
EPFL / LASEC

LASEC

Contents

1 Project description

The goal of this project was to implement a quiz generator, a new tool for teaching LASEC courses. We want to create a platform that consists in two main parts. The first one is public and aimed at students to train on the quizzes from previous years courses. A quiz consists in n questions (generally $n = 10$) with m possible answers (generally $m = 4$) and exactly one right answer. The users can choose a course and one or more chapters of this course and then generate a random quiz based on these choices. There are some more features such as a difficulty filter based on the success rate of a question. The student can also log in with his EPFL Tequila account[1]. This will keep track of all quizzes he did. He will be able to do them again, view the corrections or see some average grades.
The second part of the project is an administration zone that allows the teacher or his assistants to create courses and chapters and then simply upload Latex source files of old quizzes in a course chapter. Questions will be extracted and stored in a MySQL database. There are also a lot of features such as modifying questions, deleting or moving categories and courses, resetting questions and user statistics.
To render LaTeX in a web page, we use the MathJax library[2] for maths and we can set rules in the admin zone to convert text-formatting commands to HTML (not handled by MathJax).

[1] `http://tequila.epfl.ch`
[2] http://www.mathjax.org

2 Implementation

2.1 Files and organization overview

The folder of files consist in three parts.

- At folder root, there are all files publicly available to students. These files handle the rendering on screen, like *quiz.php* that displays a random quiz after having made the course and categories choices in the *choices.php* page. There is also the *account.php* page that displays all user statistics.
- The second part is the admin folder. There are pages that allow you to modify a course, a category or a question.
- The third one is the biggest and the most interesting one. This is the lib folder where resides all library files and PHP5 object-oriented classes used by the two parts above.

To have an easy way to configure the project as you want, all options stands in one place, in the *config.inc.php* file. There are more than fifteen options such as the max length allowed for strings, the number of questions in a quiz, the maximum number of saved stats per student, the choice of MySQL database and tables names, ...
Database login informations are sensitive, so they are not stored in this file, but in a separate one (*param.inc.php*). This allows us to restrict the access to this file at the minimum.

Below, you can see the complete tree structure of project files.

```
+-web/
  +-account.php
  +-choices.php
  +-index.php
  +-login.php
  +-quiz.php
  +-quiz_correction.php
  +-top.php
  |
  +-admin/
  | +-category.php
  | +-course.php
  | +-delete.php
  | +-index.php
  | +-question.php
```

```
| +-rename.php
| +-tex2html_editor.php
| +-top.php
|
+-lib/
| +-config.inc.php
| +-CourseManager.class.php
| +-grade.lib.php
| +-id_encoder.lib.php
| +-latex.lib.php
| +-param.inc.php
| +-parser.lib.php
| +-Question.class.php
| +-QuestionManager.class.php
| +-UserStatsManager.class.php
| |
| +-tex2html/
| | +-default.txt
| | +-tex2html.txt
| |
| +-lib_tequila/
| | +-[ ...Tequila files... ]
|
+-MathJax/
| +-[ ... MathJax 2.1 library files ... ]
|
+-images/
| +-[ ...all images... ]
|
+-styles/
| +-styles.css
|
+-tests/
| +-[ ...bunch of tests... ]
```

2.2 Parsing LaTeX quizzes

The structure of a LaTeX question is :

```
\question[<solution index>]{A question}
{Answer 1}
{Answer 2}
{Answer 3}
{Answer 4}
```

The parser library (*parser.lib.php* in lib folder) works using the following steps :

1. Remove comments starting with a %.
2. Remove comments surrounded by \begin{comment} and \end{comment}.
3. Find the position of next "\question[".
4. Extract next integer (the solution index) and match the closing bracket.
5. Get next balanced brace brackets (the question).
6. Get then n next balanced brace brackets (n answers), until a closing brace bracket isn't followed by an opening one (white spaces or carriage returns ignored).
7. Store the question, the answers and the solution index in the database.
8. Restart at step 3 until the end of the file.
9. Restart at step 1 until we have no more file to analyze.

If an error occurs at some step during the parsing, the malformed question is ignored.

2.3 LaTeX rendering

To render LaTeX in a web page, we use the MathJax library version 2.1. This is fast and really powerful for displaying math in HTML format (it uses MathML, or images depending on the browser). This is a JavaScript library and is optimal in the way that it loads only the content needed to display the current page.
However it is not able to convert text-formatting commands like \emph{} or \textbf{}, so we have a special page in the admin zone (*tex2html_editor.php*) where you can modify rules about the conversion of these commands in HTML format. Theses rules are stored in a simple text file, but their consistency is checked before saving. As an example, if a LaTeX command needs arguments (i.e. contains opening and closing brace brackets), we need to match the content inside an HTML tag, so we need to add the special marker $$$ where this content should go.
Some good example rules are :

```
\emph{}          ::        <em>$$$</em>
\textbf{}        ::        <strong>$$$</strong>
\#               ::        #
\dots            ::        &hellip;
```

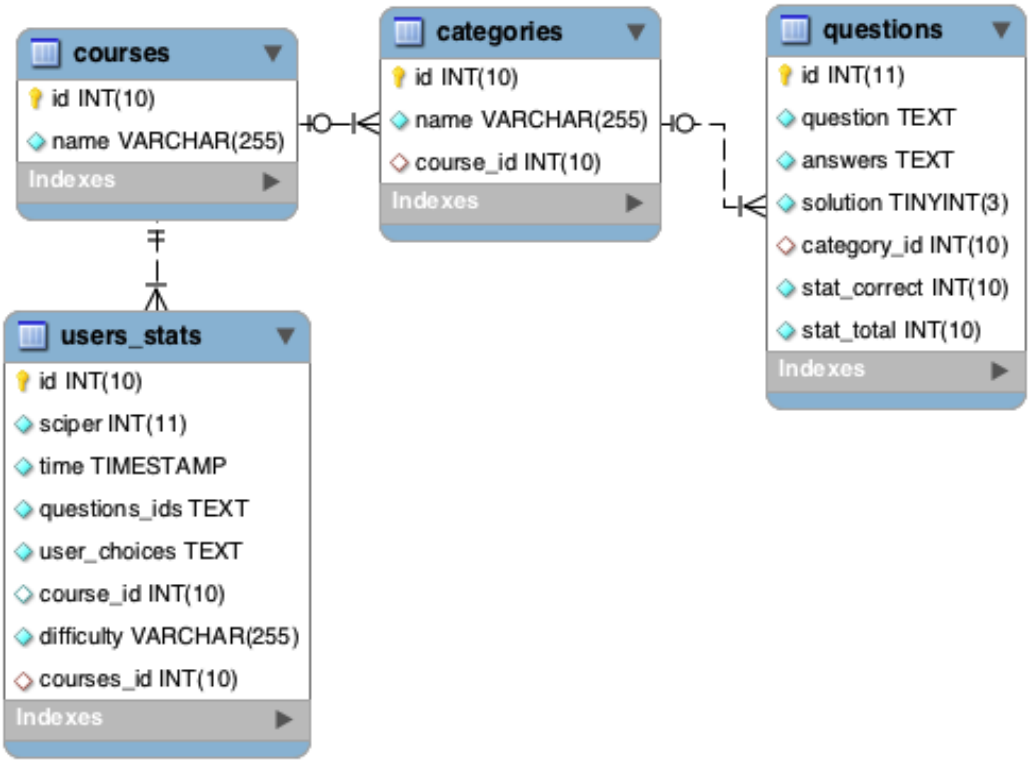

Figure 1: Structure of MySQL tables of the database

2.4 Object-oriented structure and MySQL management

The only data structure we use is to represent a question. So we have a question class in *Question.class.php* file. As you can see in the database structure of Figure 1, a question contains an id, the question statement, the m answers, the solution index, the id of the category, and statistics. You will remark that all answers are stored in a single text field, this is due to the serialization[3] of the answers string array. To be the most generic as possible, we don't fix the number of answers. Serialization is therefore the best solution to store an undetermined number of elements.
We use a MySQL database to store all datas. To avoid opening and closing connections in an disrupted way, all connections and queries are encapsulated in manager classes. Construct methods open the connections and destruct methods close them.

[3] `http://php.net/manual/en/function.serialize.php`

A goal in this project was to build a nice API that could let future projects build on this one. Here you can see all methods and public fields of the question class and the managers classes :

```
class Question
{
public $html_mode = HTML_MODE;
function __construct($title = "",
function is_correct($solution)
function audit_errors()
function html_form_field($index, $random = false)
function html_correction($index, $user_choice = -1)
function set_title($title)
function set_answers(array $answers)
function set_solution($solution)
function set_id($id)
function set_cat_id($cat_id)
function set_answered_correctly()
function set_answered_badly()
function set_stat_correct($stat_correct)
function set_stat_total($stat_total)
function reset_stats()
function get_title()
function get_answers()
function get_solution()
function get_id()
function get_cat_id()
function get_stat_correct()
function get_stat_total()
function get_stat_percent()
function get_errors()
function get_latex()
function __toString()
static function stat_cmp(Question $a, Question $b)
}

class QuestionManager
{
function __construct()
function already_exists(Question $question)
function insert_question(Question $question)
function check_ids(array $ids)  // returns true if all ids exist
function delete_question_with_id($id)
function delete_questions_from_category($cat_id)
function move_questions($from_cat_id, $to_cat_id)
function update_question(Question $question, $ignore_audit=false)
function get_question_by_id($id)
function get_questions($rand=false)
function get_category_for_question($id)
```

```
function get_questions_by_cat($cat_id, $rand=false)
function get_questions_for_cat_array(array $cat_ids)
function __destruct()
}

class class CourseManager
{
function __construct()
function course_id($name)
function course_name($id)
function create_course($name)
function delete_course($id, $delete_categories = false)
function rename_course($id, $new_name)
function get_courses_ids()
function category_id($name, $course_id)
function category_name($id)
function course_from_category($cat_id)
function course_name_from_category($cat_id)
function create_category($name, $course_id)
function delete_category($id)
function rename_category($id, $new_name)
function move_categories($from_course_id, $to_course_id)
function get_categories_ids()
function get_categories_ids_from_course($course_id)
function __destruct()
}

class UserStatsManager
{
function __construct()
function insert_stat($sciper, array $ids, array $user_choices, $course_id, $difficulty)
function get_stats($sciper)
function reset_all()
function delete_older_than($months)
function __destruct()
}
```

The spirit of the project can be seen in the API of this four classes. We have a lot of functions that exists only to check consistency or try to detect errors. Organizing the code in such a way let us have a systematic approach and we are sure to handle every single case that could occur even if it is an unlikely one.
As an example, before displaying or saving a question, we call the *audit_errors()* function that checks the consistency of all fields (at least 2 answers, non-empty question or answer, solution index in answers range, ...). This will return the number of errors found and we can act in consequence. But this is not the only thing that this function does, it will also record all

errors. We can then access this list with the *get_errors()* function and display them in the admin zone.
With this access to the database in the *QuestionManager.class.php* file, it makes it easy to check if a question already exists. This is what we do when new questions are submitted in a category. When we move questions from a category to the other, we also check if questions already exist, if this is the case, we merge the statistics.
This project contains a lot of features like that where we double check what we are trying to do. When merging courses we also verify if there are conflicts in categories name, all deletions are prompted except if the course or category we are deleting is empty, ...

2.5 Reference encoding

To be able to have a reference field that allows the user to do a specific quiz, we need a way to encode the list of n question ids into a reference.
The chosen method in *id_encoder.lib.php* consists in two steps :

1. Concatenation of the ids into a big integer. To know where to cut them later, we add the number of digits of the next id before it.
   Example : 14-7-231 will be converted into **2** 14 **1** 7 **3** 231.

2. Expressing this big number in a chosen basis. In *config.inc.php*, you can set a string containing as much different characters as you want. They will be the symbols with which your reference will be encoded with. The length of this string is the basis.
   Then using the BC native library of PHP[4], we can do integer divisions or modulo calculations with arbitrary big numbers to express our big integer in this new basis.
   Example : a quiz reference looks like this *r4hFwCbZTfM74hTazR4ja9P* for a case-sensitive alphanumeric basis.

2.6 Different difficulty levels

To generate an easy or an hard quiz, we need a greater probability of being displayed for a question without being deterministic.
For the easy mode, we will first compute the success rate in percent of every potential questions. Then, we remove the minimum rate to all questions. For example, if statistics goes from 60% to 95%, all values we will use will be between 0 and 35. We also add 5 to every values to avoid having a probability

[4]http://php.net/manual/en/book.bc.php

of zero for some questions. Now we build a new function (like a cumulative function in probability) by setting his first value to the probability of the first question (between 5 and 40). The second value, will be the first probability plus the second one. The third one will be the sum of the first three probability, ...
Now we can think about this new function as a stair function with unequal steps. By taking a random value between 0 and the maximum value of this stair function, we have a higher probability to fall on the corresponding step of a high probability question. We find this step by iterating over this stair function as long as the random value is smaller than the current step. The final value of the iterator corresponds to the question we will pick.
For the hard mode, this is exactly the same process except that we reverse the probability of the questions. Instead of removing the minimum to each percent value, we take the maximum and remove the current question statistics in percent. With this method, the question succeeded at 95% (if it is the maximum) will have probability value of 0 plus the five that we add to avoid zero probability. A question with 60% success rate will have $95 - 60 + 5 = 40$ probability value to add in the stair function.
Normal mode is just completely random questions.

3 Demonstration

Now that all features have been explained and detailed, we can try it. A demonstration of this project is hosted on a personal web server. You can access it using this link : `http://home.farquet.com/quiz_generator/`.
The administration zone isn't password-protected, so you can modify whatever you want. You are encouraged to test some features.
Note that the Tequila authentication only works if the request comes from the EFPL subnet (**.epfl.ch*). So the demo hosted on this server forces the user to be logged in with a SCIPER number set at 999999 for everyone (username and password won't be prompted).

4 Conclusion

On the paper, this project could be quite simple, because the only requested feature was the ability of parsing LaTeX files and generating random quizzes with that. This is exactly what this project can do and it can do it really efficiently (parsing lots a files takes a few milliseconds).
However, starting with this project I decided to do it properly because I knew that designing a nice API and organizing my job the best way will let me add more and more features without encountering major problems. This is how I was able to add Tequila authentication with account page, difficulty filters, question statistics, multi-upload, fancy reference encoding, all-errors handling, ability to re-take a quiz or view a correction. Every day working on it, I had a new idea, something that I could do better or a new feature to add. The only limit was time, so I had to make some choices to finally stop with this 4000 lines project.
What I was really concerned about is that my work would not only help the student to learn, but also the teacher to teach! Having a complete statistics system let see how and when students work, but most of all you can see what they didn't understand by looking at the success rate by question. This is why I added a button that let you sort the questions by success rate in the admin zone. In a few seconds, you can go in a category and see the most failed topics and then eventually clarify the subject in class.