Web-based student data collection and assessment

Thomas J. Bensky, California Polytechnic State University, San Luis Obispo

Tom Bensky, Ph.D. is an Associate Professor of Physics with interests that include using the latest web technologies in teaching.

Abstract

Gathering information from students for learning assessment is a critical need of any teacher. In this paper, the tech-savvy instructor is introduced to basic techniques of creating custom, web-based, student-data collection software. Despite widespread deployment of generalized course-management systems, the author argues that small, ultra-customized data collection methodologies, can be of great help in collecting and assessing student input, reducing demands on paper, and easing the familiar grading bottleneck while satisfying an individual's pedagogical needs.

Introduction

A stack of papers, red pen, nagging guilt, and a whole afternoon spent grading are the hallmark of a typical teacher's schedule. Machine assisted grading perhaps began with input-limited bubble-forms[1], which has since exploded with the integration of technology, notably the Internet into education. "E-assessment" systems, such as Blackboard, Desire2Learn, TurnItIn, Quia, "clickers"[2] and many others have existed for years. Since teaching tends to be highly idiosyncratic, no single system will ever completely satisfy an instructor, and certainly not a whole department or entire university. This article has a two-fold intent. The first is to show the tech-savvy instructor how to implement their own, ultra-customized, web-based, student data-collection system that can handle just about any data collection and processing task(Haptonstahl, 2008). Second, once a class set of data is available, the instructor is then introduced to simple database commands that can be used to generate custom reports based on the data.

Motivation for this work comes from an ongoing need by the author in his own teaching duties. In freshman-physics, there is a standardized "diagnostic" exam called the "Force Motion Concept Evaluation" (or FMCE short)(Thornton, 1998), which tests students on their basic knowledge of Newtonian Mechanics. The exam is designed so pre- and post-course scores can be compared for "learning gains."(Ramlo, 2008) (Such exams undoubtedly exists for other disciplines.) The FMCE consists of 47 multiple choice questions, with choices ranging from A-J. Using this important exam has been problematic for a several reasons. First, the 10-letter choices (A-J) require a special "bubble form" that has been difficult to grade using locally available Optical Mark Recognition (OMR) machines. Second, even the convenience of OMR grading still results in a stack of one-time-use papers that need additional processing (recording grades, etc.). Third, we desire quick results, so students may have a near-immediate measure of their existing knowledge of physics, as the class begins. Lastly, for the purposes of later analysis, we need individual student answers for each question. The web-developed system described below handles the entire deployment of this exam, including grading, and grade-book compilation.

The software development discussed here dubbed "web-development," is unlike that for software to be "installed" on a computer (and invoked perhaps by a double-click on an icon). Here, the software resides on a server (a computer), which is always "on" and connected to the internet. The software must be ready to run at any time, at the request of a user somewhere in the world using a web-browser. There are a multitude of options for developing such software, starting with the server: Mac, PC, or a dedicated "server appliance," either personally or professionally hosted, running either Windows, OSX, or Unix/Linux. Next comes the web-server software, Apache, IIS, or one of 50+ others. Following this comes the programming language to implement the needed task; any can be used. A database is often helpful for storing user data, and one of dozens perhaps with "sql" in the name might be chosen. "Frameworks" are also available providing convenient ties between all subsystems, while allowing the programming to focus primarily on

their specific task. CakePHP, CodeIgniter, and Django(Ranwez 2009), and Ruby-on-Rails name a few. Our choices for this work are a self-hosted Linux-based server, the Apache web-server software, PHP programming language, and MySQL database[3]. The following sections will demonstrate how one may collect, store, and assess input from their students, over the web, using these software tools.

Software

To begin, the reader must gain access to a web-ready server configured with the MySQL database, PHP scripting language and the Apache web server. Many web hosting companies offer such systems for a minimal monthly fee, including a custom domain name[4]. Such a system may also be built on a "last-generation" PC (extending its useful life), by downloading and installing the Ubuntu Linux "server" (for free)[5]. So called "LAMP" (Linux, Apache, MySQL, and PHP) systems come pre-configured and ready for use by both hosting companies and the Ubuntu installation alike.

The computer obviously serves as the moderator between the teacher and the student input. The Apache web-server is software that allows web pages to be sent out to students, when they connect to the server with their web browsers (on their own, remote computers), and handles input that may arrive from them. The MySQL database is used to store student input for later processing. PHP is a flexible, interpreted programming language that is integrated into the Apache web-server. It allows for easy communication between the web pages and the database. The language can also be used to perform grading and point-assigning tasks. All of these components are free and products of "open source" development. We find the LAMP backbone to be the most straightforward and cost effective, so we use this approach.

In design, a student-input collection system consists of first presenting a student with a web page, detailing the information needed. The page will invariably contain instructions, fill-in boxes, and a "submit" button at the bottom, that the student will click to send in their results. As a result of this click, information the student provides (their name, answers, etc.) are sent to the Apache web server which in turn sends them to a PHP program written by the instructor, to process the incoming results in some required manner. As mentioned, PHP is a full featured programming language, and in use with the Apache web server, student input will conveniently appear in global PHP variables, for use by the code itself. Typically, the PHP code will insert the student input into a database and can preform automatic grading of the incoming data. Thus, a web-based student input system consists of three parts: the web-form that the student interacts with, the PHP code to process the incoming data, and the MySQL database to store the results.

Design: Student-viewable form

To begin, the instructor must decide what information they wish to collect from the students. For the FMCE, we would like a web page asking for student identification (a name and identification number), followed by a single text-box into which the student may type a 47-character "answer string" consisting of the concatenation of individual answers selected in response to each question. Thus our design will be guided by a webpage the students will be presented with. Collectively such "forms" consist of fill-in boxes, radio buttons, drop-down lists, etc. that we are all familiar with from our own internet experiences(Kennedy, 1998, Chapter 10). In our web-directory of the server in use, the following HTML code will create the needed form.

```
<form name="fmce" method="post" action="input.php">
  Name (last): <input type="text" name="name" /><br/>
  ID: <input type="text" name="id" /><br/>
  Answers: <input type="text" name="answer" maxlength="47" size="47" /><br/>
  <input type="submit" />
</form>
```

In practice, we'll create this file on the server and save it with some name such as question.html[6]. The tags <form ...> and </form> tell the web browser that the contents in between comprise a fill-in web form. The "post" method is chosen as the method of passing the student input back to the server, with the "action" of executing a PHP file called "input.php" (see below). Readable text can be seen in the words "Name (last, first):," "ID:" etc. labeling the input fields. The <input> tags of type "text" tell the browser to give the student a one line, fill-in box, into which they can type their answers. Each input field is given a different "name" as in "id," "answer," and "name" for organization and easy retrieval later.

Design: The database

As mentioned, a database will store the incoming data. In this example, we have a student name, ID, and answer string to store. Thus we'll need a MySQL table that has these three fields. Additionally, we'll want PHP to grade the incoming results against the answer key, so a number containing the count of correct answers will be stored as well. Assuming your MySQL server is configured(Dyer, 2005), run the MySQL client from the command prompt by typing mysql -uUSER -pPASSWORD (where USER and PASSWORD are your username and password into the MySQL database). This will be configured by your web hosting company or by you the first time MySQL is run. The MySQL prompt will appear, and your database structure can be created as follows (bold is what you type with the "return" pressed at the end of each line).

```
mysql>create database fmce;
Qucry OK, 1 row affected (0.33 sec)
mysql>use fmce;
Database changed
mysql>create table answer (
-> answer_id int not null auto_increment,
-> name text,
-> id text,
-> answers text,
-> correct int,
-> primary key(answer_id)
-> );
Query OK, 0 rows affected (0.01 sec)
mysql> quit
```

We start by creating a database called "fmce," and telling the system to "use" this database for the following operations. The "create table answer (" line tells MySQL that we are creating a new table called "answer," which will be used to store the individual responses from each student. The line "answer_id..." creates a an integer (int) field that will hold a system-defined, unique, auto-incremented integer that is attached each record inserted into the database (part of the SQL schema are that no two records can be identical; this field ensures this). Next we see definitions for the student's name, identification (id), answers (all of type "text"), and the integer "correct" that will contain the number of correct answers, after the PHP program grades the student input. The "primary key" line designates the variable "answer_id" as the primary identification field for a given record. The database is now ready to accept entries from the students.

Design: Accepting and Automatically Grading Student Input

The PHP file called "input.php," handles the routing of data from the web form to the database. The following file should now be created and placed in the same folder as question.html, under the name *input.php*.

```
<?php
$name = $_POST['name'];
$id = $_POST['id'];
$answer = $_POST['answer'];
if (strlen($answer) != 47)
  {
    print("Your answer string must be 47 letters long.");
    return;
  }
mysql_connect("your.server.name","USER","PASSWORD");
mysql_select_db("fmcc");
$name = addslashes($name);
$id = addslashes($id);
$answer = addslashes($answer);
$answer_key = "abcfdbbjcaaaaeeaebbgeabcbcdcceeeeeaaaaeafbdbdef";
$correct = 0;
for($i=0;$i<strlen($answer_key);$i++)
  {
    if ($answer_key[$i] == $answer[$i])
      $correct++;
  }
mysql_query("insert into answer values(NULL,'$name','$id','$answer',$correct)");
print("Thank you. You got $correct of 47 answers correct.");
?>
```

Remember the form <input> fields called "name," "id," and "answer" in the form above? The first three lines retrieve what the student typed (from the $_POST variable structure) into each of these boxes, and stores the results into the PHP variables $name, $id, and $answer (variables start with the "$" symbol in PHP). After a quick check to see that the student typed exactly 47 answers, we instruct PHP to connect to the MySQL database server, to the database called "fmce." Next we call the PHP function "addslashes" to make all input from the students safe for insertion into the database[7]. Next, with the FMCE answer key programmed directly into the PHP program, we run a short "for" loop comparing the student's answer-string with the answer-key, counting correct answers[8]. We end by inserting their name, id, answers, and number of correct answers into the database using the PHP "mysql_query" function, followed by a a quick message to the student. This completes the entire design.

Design: Implementation and Use

Administering the exam is straightforward, with a few scenarios. First, the students can be asked to take the exam within a short duration of time relative to the first class meeting. Second, the class can be held in an on-campus computer lab for dedicated test taking session. Lastly, we have begun to notice that nearly 100% of our students now come to college with their own laptop computer. It should begin to be possible to ask students to bring their laptops to class, for a dedicated and controlled diagnostic-test class-session.

Suppose that an entire class has just taken the exam. Armed with the power of SQL in generating reports from the data in the database, we can return to the server run some

queries. For example, to view all of the raw input, we can issue the following MySQL command: select * from answer;(Fehily, 2008).

```
mysql> use fmcc;
Database changed
mysql> select * from answer;
answer_id,name,id,answers,correct
102,last15,first48,275,cjbdfhibbceeeihhhgbjabeajebifgjifabbhacidhdhfae,3
103,last27,first76,6157,chhhigbjfhfefbfhcdifcbcfjhgeadadbhajecjajeffgbd,5
104,last89,first42,1810,dgdgbddihechhdfhdajcajhgfdhicbaghccjhgicaacidif,4
105,last72,first86,5626,jjfgfaadjcfjiccbcbhagiaigegcbfhbediadiedajcjcha,5
106,last51,first86,8234,feggddbafdgdfbgdbabgdcfgbhdcccaihhfaahbfahjfifj,9
107,last6,first65,524,gaccgcaagdcgbadhbeecjfaibieajbbfbdiihijebcbdcea,3
108,last33,first90,5488,fibfhcecdddejfhidfgcahebbhgcafhgdicbbgeeahiadfi,4
109,last71,first18,5297,jbdedecjgceejicbadieijcgjfcicddcfhhibjiibccabfc,4
110,last12,first95,601,fhjhdjcfiejchfjeebdcjffcgghihgjdeiaiidegidifjij,7
111,last43,first91,2773,hiicaejidhfcajbahjdbgcffhfdgjcjgahjbbijefehfeig,4
10 rows in set (0.00 scc)
```

In the output, we see 10 entries (of our mock student input). The auto-incremented integer field (answer_id), students' names, ID number, answers, and number of correct responses are clearly visible. Other examples of useful SQL commands are as follows.

- To find the average number of correct answers and the standard deviation: select avg(correct),stdev(correct) from answer;
- To find all students who answered 'a' for question #1: select name,answers from answer where mid(answers,1,1)='a';
- To see a histogram of how students answered question #10: select mid(answers,10,1) as choice,count(*) from answer group by choice;
- To see the grades sorted from highest to lowest: select name,correct from answer order by correct desc;
- To see the grades sorted by student name: select name,correct from answer order by name asc;
- To see a histogram of the correct answer grades: select correct,count(*) from answer group by correct;

Design: Reflection and Improvements

A computer-mandated deadline to input one's answers is an obvious addition. To do this, the current date must be checked against a set deadline date. PHP is well suited to handle date comparisons as follows. Suppose the deadline of 5:20:00 pm on November 19, 2010 is required. This can be converted to an integer "time stamp"[9] using the PHP mktime function as in

```
$deadline = mktime(17,20,0,11,19,2010);
```

while the current time of day can be found from the PHP time() function. We collect this code and place just below the <?php tag in the input.php file above as

```
$deadline = mktime(17,20,0,11,19,2010);
if (time() >= $deadline)
  {
    print("Sorry. The deadline has passed.");
    return;
  }
```

where the if statement is used to see if the current time (time()) is larger than the deadline time ($deadline). If so, the deadline has passed and the student is not allowed to proceed.

Conclusions

We demonstrated how to create a small, "ultra-customized," web-based, student input collection system that completely automates the administration of a 47-question physics diagnostic exam. Grading is automatically performed, provides instant results, and performs grade-recording functionality. No existing online system can handle this task in such a straightforward manner. When data collection is needed, we encourage the reader to look beyond larger, more generalized course management suites, in favor of smaller, ultra-customized data-collection systems as outlined here. The LAMP components outlined here allow for endless experimentation and fine-tuning for virtually any online data-collection need.

The author wishes to thank A. Peckham for many helpful discussions about MySQL and the anonymous referees for their helpful suggestions.

Endnotes

[1] Scantron forms have been in use for approximately 30 years.

[2] See http://www.blackboard.com, http://www.desire2learn.com, http://www.turnitin.com, http://www.quia.com, http://www.einstruction.com, and http://www.iclicker.com.

[3] See http://www.apache.org, http://www.mysql.com and http://www.php.net.

[4] See http://www.godaddy.com, the "Economy Plan," for $5.00 (USD) per month.

[5] See http://www.ubuntu.com/

[6] See http://atom.physics.calpoly.edu/web/question.html.

[7] So called "SQL injection attacks" must be guarded against carefully.

[8] It is impossible for a student to see the answer string. PHP programs reside on the server; only their output is available to the connecting web-browser.

[9] The Unix timestamp, or the number of seconds that have elapsed since January 1, 1970, is an odd time metric, but is extremely useful for date comparisons to within one second.

References

Dyer, Russell. MySQL in a Nutshell. Sebastopol, CA: O'Reilly, 2005.

Fehily, Chris. SQL (3rd edition). Berkeley, CA: Peachpit Press, 2008.

Haptonstahl, Stephen. Web-Based Data Collection with PHP and MySQL. The Political Methodologist, 15(2), 11-16 (2008).

Kennedy, Bill and Musciano, Chuck. HTML & XHTML: The Definitive Guide (6th edition). Sebastopol, CA: O'Reilly, 2006.

Ramlo, S. Validity and reliability of the force and motion conceptual evalution. American Journal of Physics 76(9), 882-886 (2008).

Ranwez V, Clairon N, Delsuc F, Pourali S, Auberval N, Diser S, Berry V. PhyloExplorer: a web server to validate, explore and query phylogenetic trees. BMC Evolutionary Biology. 9(108) (2009).

Thornton, R.K. and Sokoloff, D.R. Assessing student learning of Newton's laws: The Force and Motion Conceptual Evaluation. American Journal of Physics 66(4), 228-351 (1998).