Group 40 Portfolio 2

Scott Mueller & Sam Eue

Our project is an online trivia game that runs on an Apache server and uses a MySQL and PHP backend.

Contents

1 Overview

The idea of this project was to simplify and speed up a small informal trivia game at club event for ISU CXC. The idea was to host several panels at club events and this system could be used to simplify the process of setting up and running of trivia panels. We wanted to allow players to use their phone to enter in answers for various trivia questions. When this product will actually be used, it will be running on a closed network. Players will connect to the network and will be cut off from the internet to avoid cheating by looking up the answers online. Since all the players are intended to be in the same room, the host will have the questions displayed on a projector. Players will connect to the server and see a page for them to enter a username. This will be stored in the MySQL database along with their IP address so we may uniquely identify the players as well as the player's score. After all the questions have been gone through the host of the game can walk through the answers on the projector and then be able to display the top 3 overall players in the game.

2 New and Complex

2.1 New

In this project we made use of an Apache web server running on a Raspberry Pi. The Pi includes a PHP interpreter and uses a combination of JQuery, PHP, and HTML to create web pages. Since we wanted this to run and display nicely on a mobile phone, we made use of the Twitter Bootstrap framework. Bootstrap provides a mobile first design to web pages ensuring that what we created would look good on both a desktop browser and a phone browser.

The Pi also runs a MySQL server and uses PHP to query the database and provide data to the web pages. To reduce the number of calls to the MySQL server, we installed a program called Memcached. Memcached allows us to set variables in the server's memory for use later by any client on the server. Querying for a memcached variable requires less time and processing than querying a MySQL database.

In order to create the tables for the database we created SQL scripts. The script creates 2 tables, one for the player data with columns for the player's IP address, username, and score, and another for the questions with columns for the question, answer, and category. To enter all of the questions to the database we used Java JDBC. The provided jar file when executed will connect to the specified database with the provided username and password then parses the provided text file with the questions to use and enters them into the database. Detailed instructions for use of the jar are provided in the included readme.

2.2 Complex

This project makes extensive use of SQL within PHP code. When the player's first connect, we need to record them. This is done via a JQuery script which calls a PHP script with the player's username and loads them into the database.

Every time the host loads the next question, we make a call to the database to get the question, answer, and section. These values are then loaded into Memcached variables along with the current question number for each of the players to retrieve. This is done by a JQuery script on the player's side that queries the memcached question number against the question number stored in the player's session data. If the numbers are different, we load the new question to the player's screen from the memcached data.

When the player submits an answer for the question, we again use JQuery to call PHP which compares the player's answer against the memcached answer. Since people may not be exact on spelling, we check to see that the strings match by at least 80% using the built in PHP function. If the answer is close enough, we award the player a point and update their score in the database.

Finally, when the game is over we use JQuery to call PHP and query the database for the 3 players with the highest score and display them on the page for the host to show.

3 Bloom's Taxonomy

Synthesis/Creation, Analysis, Evaluation

Memcached:

For this project we had to devise a way to update a variable on the server on the host page and allow the players to access it. Originally we had the host set a variable in the database and players would open a new connection to the database to check the variable. However, we became concerned over the overhead of the connections the players were making to the database. We also wanted to query every few seconds to verify that we haven't moved on to the next question, however the amount of time it would take for an SQL query to complete did not allow for this.

This led us to memcached variables, a way to save global data on in the server's local memory. This way we didn't have to keep opening connections to the database and made it so that we only needed to open one connection per question. When the host moved to the next question, they would update a memcached variable with the current question number. The database would be queried for the question and answer and the values would be stored in memcached variables. Since Memcached queries complete in less time than queries to a MySQL database, this allowed us to pull on the player's end for updates at the rate we wanted to. After a rough patch of refactoring the code and database it seems to work quite well for our application.

Spell checking:

We needed a way to check if the typed answers of the player were close to the stored value without being too strict or lenient. After some digging we found a PHP function that will compare the two strings and outputs a percent similar value, we decided to cut off the answer if it

is has than an 80% match. This comes from a set of tests we did comparing different strings using the method and deciding upon an acceptable level of misspellings.

JavaScript - AJAX and jQuery:

We also used AJAX and jQuery to simplify the JavaScript functions to allow for easier readability and debugging. The use of AJAX allows us to make calls to PHP functions without having hang ups on the client side. This caused some issues with the memcached because we needed the values saved from the PHP before we updated the information on the page in a different PHP function. In short we needed to make AJAX class synchronous to other AJAX calls while still having the webpage be responsive. JQuery made accessing DOM elements much easier since most of what the JavaScript is doing is updating text elements and attaching onclick handlers to DOM elements. With the combination of the simplified AJAX and jQuery calls we have the ability to make the code much cleaner and understandable.

Bootstrap:

We also elected to use Bootstrap to make the pages look consistent for whatever device is loading the page. This is because we intend for the game host to be using a projector for displaying questions and we expect the players to be using some sort of portable device such as a phone or tablet to provide their answers. While there are methods with which to do this using traditional CSS, it requires a more advanced knowledge of CSS classes than we have. It can also be difficult to style CSS to display properly on screens of different sizes. By using Bootstrap we

found it takes little to no effort to have the same page display intuitively for any sized browser, a much needed assist with all that is needed to complete this project.