3/16/18

AAW Site Search Engine Built

This search solution is built in following scheme.

1. Build database table that will contain each page content and URLs in each row.
2. Take input for keyword search in search box.
3. Input keyword search is limited for 33 characters for security reason.
4. Input keyword search is tested for blank input.
5. Input keyword search is secured for injection attacks via trim(), strip\_tags(), and htmlspecialchars() PHP functions. A custom function in the codes will do this.
6. Input keyword is checked to see if it’s a phrase containing multiple keywords.
7. Single keyword and a phrase are handled differently via switch case.
8. A phrase will call custom built search function in the codes attempting to search match for whole phrase and breaking down keywords via spaces with PHP explode function and apply search function with a loop.

No Good Solution on Internet

I scraped internet in search of a “simple” custom local-level site search solution. At least 10 solutions exist as archives in internet junkyard, but none are maintained and upgraded as of about years 2004—2005. They were all written in old PHP or PERL versions without security upgrades. Original authors abandoned them. Current PHP version is 7. Most old PHP site search solutions were not even in version 5 but version 4 and security issues were not addressed in the codes. I tried studying them to modify and upgrade the codes, but I realized it would take much time to decipher these undocumented packages.

Apache Server Shortcoming

I am also disappointed with Apache server progress. I have known Apache http server for almost 17 years. I became master of it about four years ago. I thought someone in Apache Foundation would have contributed a module that provides a site search or server side search solution by now. Despite all these proud years in Apache server’s reputation, although there is a module that iterates through directories to search for specific file names and displays them on html page, there is no module at all that would search keywords within the files and display the file names and its location.

Demise of Google Site Search Solution

Many websites on the internet are using google search to integrate into local site search through some JavaScript solution that Google have provided for years. This JavaScript solution is very crude and not customizable for its appearance. I dislike JavaScript for several good reasons. Obvious reasons are that most people block JavaScript on their browsers and that JavaScript solution like Google site search acts like a framework that calls to unknown codes written at the Google server location. The websites that use them will have their lives at the mercy of various conditions-- availability and control at the source of those frameworks. Not surprisingly, I read recently that Google will end site solution service at April 1, 2018, <https://searchengineland.com/google-site-search-way-now-271366>. Many government websites, important sites, and especially public school sites are using this Google search feature. For example, <https://www.buffaloschools.org>. They are all in big trouble.

Enterprise Multi-million cost Level Tools

I spent about one whole week exploring solutions in no-SQL world. This simple site search need with little simple codes of program that I can imagine easily in my head is very scarce in available resources in practical world. I got absorbed into modern search engine methods that use combination of SQL database and so-called no-SQL systems. Most popular ones are Apache Nutch, aka Apache Solr, aka Apache Hadoop, and Elastic Search. It was all intellectually fascinating experience to explore those solutions, but I have not a luxury to build the complete infrastructures to deploy those. I learned a lot in trying to build those systems, but I had to abandon them. I didn’t have time (I realized) and the solution I need would be an over-kill with those systems. Yet I learned that no-SQL systems really were not SQL free. Although they store their data files in XML format files in the server in regular directory, the system needs to store its “index” or meta data information in SQL databases in order for this system to work. Again, those no-SQL systems require time consuming IT engineering tools to install and configure at server level for infrastructures.

Real Independence Local Site Search Engine

Many shallow web developers only have the knowledge of deploying ready-made packages. Search solutions usually come with those ready made packages, such as oscommerce, wordpress, PHP Nuke, PHP BB, .Net Nuke, Megento, Bootstrap, and many others. There is no easy good search solution for real custom-built website. People are saying, “don’t reinvent the wheel” in the internet forums, but I realized finally that I had to code from scratch for this one. I scraped all the packages at Github and found nothing good too. Most attractive package I came close to using was “tipusearch”. I analyzed it and realized it’s built like Google site search solution with a framework style, which is calling and interacting with JavaScript codes or other files at various locations across the internet. The coder make the package obscure and much complicated and lengthy than necessary. I decided not to use it especially because of the security reasons and my prejudice against lengthy JavaScript packages. Try to analyze those JavaScript files and codes in the Tipue package. Codes are not nicely lined in files and blocks are not nicely placed to understand. Just rearranging the lines and blocks of codes in files to make them readable will take days. No thanks Tipue! Thus, I have completed building my own search engine from scratch. This whole project with all activities I described above took me nearly two months. I didn’t work for myself full time for two months. I worked on this project on my spare times. I was also working for other people with my IT business activities.

<?php

/\*Author: Christopher McGrath.

\* AAW Web site search v1.01.

\* mcgrath\_ch@hotmail.com.

\* All rights reserved.

\*/

//session\_start();

//$keyword= "mathematics"; // this will come from post

//$keyword= $\_POST[keyword];// original line is enforce with security measures below

if (empty($\_POST['keyword'])){

 //echo 'Enter a key word other than blank to search.';

 $keyword\_size=0;

}

else{

$keyword\_secure= secure\_input($\_POST['keyword']);// pass input keyword to function named secure\_input to clean up the input.

$keyword\_secure= htmlspecialchars($keyword\_secure,ENT\_QUOTES);// weird case. htmlspecialchars function needs to be applied twice--

//one in function and one out of the function. Otherwise, it doesn't strip the html chars.

//$keyword\_secure= stripslashes($keyword\_secure);

$multi\_keyword = explode(" ", $keyword\_secure);

$keyword\_size=Sizeof($multi\_keyword);

//----------------go to switch from here-------------------

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

The htmlspecialchars() function converts special characters to HTML entities.

This means that it will replace HTML characters like < and > with &lt; and &gt;.

This prevents attackers from exploiting the code by injecting HTML or

Javascript code (Cross-site Scripting attacks) in forms.

stripslashes() removes back slashes.

 trim($data) will trim specified chars

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function secure\_input($data) {

 $data = trim($data, "%\_\\/"); //This will trim all those chars inside " ".

 //Don't need stripcslashes with while loop below anymore.

 $data = trim($data);// apply trim again for white spaces before and after the keywords

 $data = strip\_tags($data);// strip html tags

 $data = htmlspecialchars($data, ENT\_QUOTES);

 //while(strchr($data,'\\')) {

 //$data = stripslashes($data);

 //}

 return $data;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function database\_search($keyword\_to\_search){

 include './data/credentials.php';

 //connect to database

 $conn = mysqli\_connect($database\_host,$usr,$pwd,$database\_search)

 or die("Connection failed: " . mysqli\_connect\_error());

 $keyword\_secure2 = mysqli\_real\_escape\_string($conn, $keyword\_to\_search);// security measure.

 //Cleans the var of chars used in SQL statement [NUL (ASCII 0), \n, \r, \, ', ", and Control-Z]

 $sql="SELECT url FROM page\_file WHERE body LIKE '%". $keyword\_secure2. "%'";//build query

 $result = mysqli\_query($conn, $sql) or die(" Query failed: " . mysqli\_connect\_error());

 mysqli\_close($conn);

 $count = mysqli\_num\_rows($result);

 if($count<1){

 $output = 'No match found!';

 }

 else{

 while ($row = mysqli\_fetch\_assoc($result)){

 $url = $row['url'];//['url'] is field name in database

 $output .= 'Found page URL: <a href=" '.$url.' ">' .$url. '</a><br>';

 }// end while

 }//end else

 return $output;

}// end of function database\_search

//$keyword= metaphone($keyword);

?>