Guidelines for Building a Document Sharing System
Jamison Bryant
Explorer Post 1010

Guidelines for Building a Document Sharing System

1. Introduction

One of my biggest frustrations as I have participated in Botball is how difficult it is for my team and I to share files. I am homeschooled and everyone else on my team is in public or private school. While they can see each other all day long, I only get to see them at the once-a-week Botball meetings. As GCER drew closer, we began to meet more frequently and for longer. But the problem of not being able to communicate outside of meetings remained. As a result, I decided to build a document sharing system (DSS) for my team. While we are using this for sharing documents, it could be used to share any type of files.

The benefits of a file sharing system are that any team member is able to access any of the files any time, if they have a username and password. This system was successful because t solved a major problem for my team. We now had a way to share documents outside of meetings (other than by email). Now, instead of emailing a certain file to everyone one the team, I can upload it once, and everyone can download it to their computers.

2. Preliminary Designs

The first thing that I did was define the requirements for the DSS. The requirements included features such as a way for users to open the files, download them, update and then upload them. I represented it like this:

KrashWeb : Post 1010 Robotics : Internal Competition
Document Library
• Document1 (Open | Download) • Document2 (Open | Download) • Document3 (Open | Download) Report Abuse

At the top of each screen is a blue banner with the users' title at the top of the page. Below that is the DSS page name, so that the users could keep track of where they were. Below that is a list of documents (or any files) that are available for either opening or downloading, with a hyperlink for each respective activity. I included a link that would allow users to report abuse, for example, a user uploading his/her entire iTunes© library.

After some thought, I decided that this wasn't necessary. I was simply not going to allow users to upload files at all, but rather than leave that solely to the administrators (me as the installer and whoever else I designate). This would give me the ability to censor each file, and make sure it contains no malware or other things that can wreak havoc on a computer. It wasn't that I didn't trust

my team; it was that I didn't trust my security system, and was afraid that someone outside the team could hack in and create problems. This is my first recommendation: **allow only administrator-level users to upload files to the document sharing system, unless you trust your security completely** (see Security Features below).

Now that the document storage design had been completed, for the place where users downloaded documents, I needed to design the screen where administrators uploaded documents:

KrashWeb : Post 1010 Robotics : Internal Competition

Administration Panel

…

File Source:

Browse…

File type:

MS Publisher 2003…

Description:

Type a short description of the file here. (optional)

Upload File

…

In this case, the body of the screen contains an input box where the user can choose the file to upload. When the user clicks the button that says 'Browse' or 'Select File' (depending on the browser and operating system), a dialog box opens that allows them to select a file, a drop-down menu where the user selects the type of file being uploaded, and a text area where the user can enter a short description of the file being uploaded.

While some may believe that these initial "rough drafts" of the application are unnecessary, I have been programming for 5 or 6 years now, and I have found one thing to be true of designing any user interface: **always make rough sketches of what you would like the application to look like; this makes programming much easier and faster.**

3. Security Features

While I realized that the drop down menu to select the file type and the document description field were not strictly necessary, I included them anyway to make uploading a document more user-friendly. In addition to user-friendliness, they increased the security of the application.

For example, the drop-down menu limits the types of files the user can upload. If the user tries to upload a file with a .xyz extension, he/she will not find that the .xyz extension is not one of the entries in the drop-down list, and therefore might decide not to upload the potentially harmful file. This brings me to my second recommendation: **never fully trust your users when they have access to site controls. Also, never trust submitted form data; always check it.**

There was a way to make my file uploading security a little better, but I decided not to employ it because of the additional effort it would take (I was fully into the GCER competition by then). Nevertheless, I recommend that you **use file filters in your document sharing system**, meaning that only files of a certain extension will be uploaded. Suppose again that the user tries to upload a .xyz file, and doesn't see it on the drop-down list, but randomly chooses an item on the list anyway. This would cause the document to upload successfully, but poses a risk for the users downloading the file, if the .xyz file is malware. However, if the program could use if-then statements to determine that the .xyz file extension was not a supported file type, it would not upload the file and return an error for the user to see. An example is pictured below:

KrashWeb : Post 1010 Robotics : Internal Competition

Administration Panel

…

ERROR: UNSUPPORTED FILE TYPE!

File:

C:\Desktop\helloWorld.xyz

Browse…

File type:

MS Excel 2003

Description:

This file sends the text "hello world" to your mobile phone!

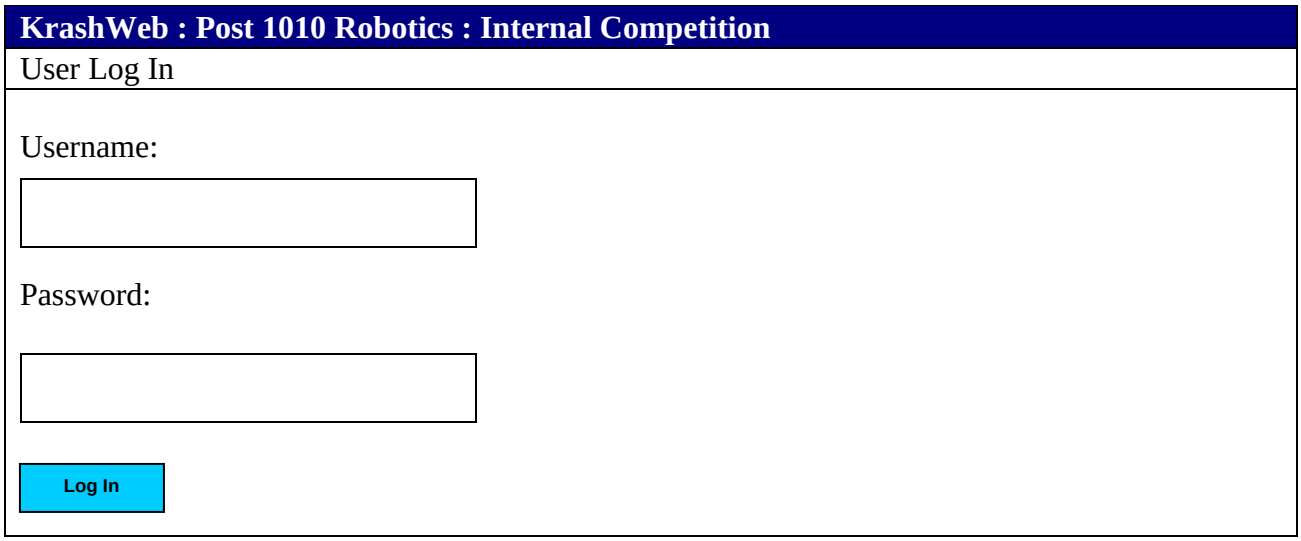
Upload File

…

In fact, I found the concept of an administration panel so beneficial that I put controls for almost all of the content on the website on this page. Of course, this meant that I would have to be much more selective about who became administrators, and I certainly recommend that you do the same.

Another essential asset of a document sharing system that is open only to a select few is a way for those select few to log in and view the documents. My document sharing system accomplished this task by requiring that the person attempting to access the site log in with a username and password, and then checked the submitted information with records in a database, which we will discuss in a moment. The log in page looked something like the following figure.

If the username and password given are legitimate, the user is granted access to the document library. If not, the user is directed back to the login page to try again. An interesting idea that some big companies incorporate would be to limit the number of times a user can try to log in to three. This would require keeping track of the number of times a specific IP address has visited the page and clicked "submit". The harder part of that would be somehow locking that IP address out after the allotted number of tries had been exhausted. This is merely a suggestion and is not a feature that I chose to incorporate in my team's file sharing system.



One recommendation that I certainly make is that you **protect yourself from SQL injection.** This rather ominous-sounding term simply means that someone exploits the fields in your login form to submit queries to your database. For example, a hacker might type in the query he or she wants to submit (perhaps a DELETE ALL) query, but then append two minus signs ("--") to the end of their query, which effectively renders the processing query that you wrote useless. The double minus sign signifies the start of a comment, and by appending it to the end of their query, the hacker tells the computer to ignore your query and replace it with his.

It is not hard at all to protect yourself from SQL injection. Depending on the language that you are using to process the form data, you can probably find a function specifically designed for removing dangerous characters and/or words from form data. For instance, in PHP the function is mysqli_real_escape_string(). I use this function frequently, and I haven't had a hacker succeed in hacking my database yet.

Another recommendation that I would make is to **encrypt your user's passwords in your database.** This is a safeguard in the eventuality that a hacker does worm his way into your database. If you haven't been encrypting your user's passwords, then the hacker can read the passwords easily. However, if you encrypt them, there is no way for him (or her) to read the passwords. Encryption, depending on the type, is very secure. For example, the word "cat" encrypted is 9d989e8d27dc9e0ec3389fc885f142c3d40f0c50. Imagine trying to decode that. Note that for a user whose password is the word "Botball", you can't search the database for a user whose password is "Botball" if you encrypt your user's passwords. Instead, you must search the database for a user whose password is the encrypted form of "Botball".

4. Storing and Retrieving Data

If one is to run an application in which one shares data, there must be a way for one to keep track of and store this data. Because my application was web-based, I opted to use databases to keep track of all of the files and the member information. We will consider two databases: one for storing file information (including a reference to the actual file) and one for storing member information, which is used by the log in page.

The file storing database stored the file name, the path to the file, the type of file, and other relevant information. One thing it did *not* store was the actual files, as they would have been stored in binary code that would have been impossible to read. Instead, the database stored the path of the file, which was then referenced by a program that loaded the documents in a database one at a time, much like you saw in the original sketches of the application that I made. It would print a link for opening and a link for downloading to the screen for each entry in the database. Of course, this would require a program for getting that information from the file uploading controls on the administration panel.

Storing member information is just as simple. The absolute minimum of information that the database needs is a username and a password for all of the users on the system. With this information, the login program can cycle through each username and its respective password and compare it to the information submitted by the login form. If one and only one record matches, then the user has entered the correct username and password, and they can see the documents that have been uploaded. If not, then the user has not entered a matching username and password, and they must be redirected to the login page as was discussed earlier. **However, for increased security, I recommend that the database contain *three* fields: username, password, and <u>rank</u>.** The rank for each user can be something like "Administrator", "Moderator", or "Basic User". In my application, only the Administrators could upload documents.

5. Miscellaneous Recommendations

There are a few recommendations I would like to make that don't necessarily have a category of their own, besides miscellaneous:

a) If you have your team and ONLY your team design the file sharing system and you host it on the Internet, I recommend that you put a link in your documentation pointing the documentation judges toward your website, with a sample username and password that they can use to log in. You never know, it might give you a better score.

b) I recommend that you use PHP to program your file sharing system. Not only is it the language that I successfully programmed my file sharing system in, and I have found that it is better than another scripting language such as ASP.
c) I recommend that you avoid duplicated code wherever possible. In my experience as a programmer, I have found that duplicated code is hard to debug and hard to maintain. It goes without saying that you should also comment your code well.
d) Finally, if you find that you do not understand scripting at all, I recommend the excellent book *Head First PHP & MySQL* by Michael Morrison and Lynn Beighley. This is the book that I learned with, and I can testify to its excellence. In fact, I would recommend the Head First series for learning any computer programming language.

6. Review

In this paper, I have given several recommendations for creating your own file sharing system. I will reiterate them here, for review and to allow me to say several words on each one.

a) Allow only administrator-level users to upload files to the document sharing system, unless you trust your security completely. This prevents malicious users from uploading files that can be harmful to your web server or your user's computer. An example of this is using the file uploading system to spread a virus, such as the I Love You virus.
b) Always make rough sketches of what you would like the application to look like; this makes programming much easier and faster. If you do this, then the designs that you go through might be able to be used by other applications. Remember, if you're including the file sharing system as part of your documentation, you want it to look great.
c) Never fully trust your users when they have access to site controls. If, for instance, the user has had no experience with programming and inadvertently crashes your system, what do you do? Only give full privileges to people who you trust and those who know what they are doing.
d) Never trust submitted form data; always check it. This is especially true of data that another script will eventually load. If somehow a worm gets embedded in your system, you can spend weeks trying to find it and fix it, as I have many times in the past. What is more, during those weeks, your team is unable to share documents!
e) Use file filters. This will allow you to control what kinds of files can be uploaded to your system, and what kinds can't. Just make sure that you don't block out all code, in case your team needs to share the robot's code.
f) Protect yourself from SQL injection. This is an essential thing to do, and it's really too simple to not do. The repercussions from this safeguard will benefit your application immensely. Besides, you'll be frustrating hackers at the same time.
g) Encrypt user's passwords. This will prevent you from having to change all of your user's passwords in addition to resetting the database in the event of a hacker breaking in. Keep in mind that you'll have to tell your script to look for an encrypted password when a user logs in.
h) If you have your team and ONLY your team design the file sharing system and you host it on the Internet, I recommend that you put a link in your documentation pointing the documentation judges toward your website, with a sample username and password that they can use to log in. You never know, it might give you a better score. (I chose to repeat this recommendation because it was one of the deciding factors in my team's documentation scores.)