

SINGLE SIGN ON SYSTEM

By

Illya bte Ridhwan

Supervised by

Mr Abdullah Sani Abdul Rahman

Dissertation submitted in partial fulfillment of
the requirement for the
Bachelor of Technology (Hons)
(Information Communication Technology)

JUNE 2006

Universiti Teknologi PETRONAS

Bandar Seri Iskandar

31750 Tronoh

Perak Darul Ridzuan

PUSAT SUMBER MAKLUMAT
UNIVERSITI TEKNOLOGI PETRONAS

UNIVERSITI TEKNOLOGI PETRONAS
Information Resource Center



IPB183585

tk

5105.588

1721

2006

1) PHP Computer Program Language

2) Apache (Computer File: Apache 2.0.40)

CERTIFICATION OF APPROVAL

Single Sign-On System

by

Illya Bte Ridhwan

Supervised by

Mr Abdullah Sani Abdul Rahman

A project dissertation submitted to the

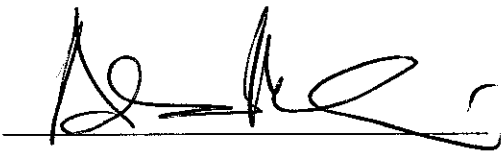
Information Communication Technology Programme

Universiti Teknologi PETRONAS

in partial fulfillment of the requirement for the

BACHELOR OF INFORMATION COMMUNICATION TECHNOLOGY (Hons)

Approved by,

A handwritten signature in black ink, appearing to be 'A. S. Rahman', written over a horizontal line.

(Mr Abdullah Sani Abdul Rahman)

UNIVERSITI TEKNOLOGI PETRONAS

TRONOH, PERAK

June 2006

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.



(ILLYA BTE RIDHWAN)

ABSTRACT

This report is provided to explain regarding the Single Sign-On system. In this report, it will give a thorough view on Single Sign-On focusing on the system purpose, scope of study, methodology, results and conclusion. For the purpose point of view, this system is a type of software authentication that enables a user to authenticate once and gain access to the resources of multiple software systems. This is to make sure that the user authentication process becomes easy as they don't have to enter multiple usernames and passwords for multiple systems. In order to achieve this objective, the scope of the system has to be analyzed first. For this system, it will only relate to the systems that are web-based applications. In other words, we can call this system as Single Sign-On Web Portal. For the methodology part, PHP language as well as Apache server will be used to complete this project. It is one of the most demanding types of programming language nowadays. This system will also be divided into 2 parts: user interface and administration interface. For the results part, this report will shown the work progress as well as the screenshot of the system interface. The discussions along the work progress will also being included. Last but not, for the conclusion part, this report will conclude all the work done and provide recommendation for system enhancement in the future. This report will be guidance through out the system, from the first it being planned until the end product comes out.

ACKNOWLEDGEMENTS

During my task in completing my final year project regarding Single Sign-On system, I had gone through various challenges and problems upon completing all the tasks and assignments that had been assigned to me. Without the help from other people, I would be in great difficulties.

As a token of appreciation, I would like to take this opportunity to express my heartiest gratitude to Universiti Teknologi PETRONAS (UTP) for giving me a chance to further my study here and grab so many experiences since my first year of study until the final year. As their mission is to provide a well-rounded student, they trained student who can stand on their own feet without being too dependent on anyone else and create a pleasant study environment to the student as well.

I would also like to thank one of my supervisors while I'm doing my Industrial Internship, Mr. Hazaruddin Zulkhillpully and my UTP supervisors, Mr. Abdullah Sani for their advice, concern and words for encouragement. Besides that, I also would like to thank you for all their effort in order to help me to finish my project either directly or indirectly. Without their help, my project won't be complete. My greatest appreciation also goes to them for being patient in spending many hours in finding solutions and helping me out with the projects. I thank them for the support, cooperation and the understanding that he had showed throughout all the activities that we had done together.

Last but not least, I would like to say thank you to all my friends regardless where are they now, either in UTP or anywhere else. Without help from you all, I don't think I can stay long here and thank you so much for all the memories we created that I will always blossom in my heart.

TABLE OF CONTENTS

CERTIFICATION	i
ABSTRACT	ii
CHAPTER 1:	INTRODUCTION	1
	1.1 Background of Study	1
	1.2 Problem Statement	2
	1.3 Objectives and Scope of Study	4
CHAPTER 2:	LITERATURE REVIEW/THEORY	6
CHAPTER 3:	METHODOLOGY	8
	3.1 PHP	8
	3.2 Apache Server	12
	3.3 MYSQL	18
	3.4 Single Sign On Mechanisms.	27
	3.5 Tools Required	28
CHAPTER 4:	RESULTS AND DISCUSSION	29
CHAPTER 5:	CONCLUSION AND RECOMMENDATION	36
REFERENCES	39

LIST OF FIGURES

Figure 3.1	Relationship between Server and Client through PHP
Figure 3.2	Relationship between Server and Client
Figure 3.3	Apache modules
Figure 3.4	Model of Web Application
Figure 3.5	MYSQL Control Panel
Figure 3.6	Create database called “sso”
Figure 3.7	Database “sso” has been created
Figure 3.8	SQL button for entering data in the database
Figure 3.9	Upload the database data entry
Figure 3.10	Upload the location of the textfile
Figure 3.11	MYSQL database successfully upload
Figure 3.12	An Single Sign-On Architecture Utilizing Scripting
Figure 4.1	System 1 Web Application Login Page
Figure 4.2	System 1 Web Application Main Page
Figure 4.3	System 2 Web Application Login Page
Figure 4.4	System 2 Web Application Main Page
Figure 4.5	Single Sign-On Web Application Login Page
Figure 4.6	Single Sign-On Web Application Main Page
Figure 4.7	Admin Control Panel Page
Figure 5.1	SAML scenarios

LIST OF TABLES

Table 5.1	Advantages and Disadvantages of Single Sign-On Architectures
-----------	--

CHAPTER 1

INTRODUCTION

For a large enterprise, as the number of passwords each user is required to maintain increases, so do the support calls. With each of these calls having an associated operational cost of some of money, and the increasing number of applications in use, businesses cannot afford the productivity lost through continuous password resets. Single Sign-On have evolved as a cost savings solution to minimize support calls, and at the same time simplify the administrative process of authentication and authorization. This also can save the administration staff time as they don't have to set and reset usernames and passwords for the other staffs who forget their account.

1.1 Background of study

There are two main types of single sign-on architecture types – Web-based and non Web-based. For the most part, the various enterprise products available can be segmented into one of these two categories. Today, non-Web-based applications are sometimes referred to as legacy applications, and therefore, the associated Single Sign-On products that they interoperate with are referred to as legacy Single Sign-On products. To have a better understanding what features are important in helping you selecting an implementing a Single Sign-On product, a deeper look is being considered at these two Single Sign-On.segments.

1.1.1 Web Single Sign-On

The Web is made up of portals which act as gateways to many layers of web sites. Some portals have a unique focus, while others try to be all things to all people. A portal is often the front-end to a lot of different enterprise applications that converge in one Web based user interface creating an organizational single point of presence. When the Web first evolved, Secure Sockets Layer (SSL) was sufficient for passing encrypted

passwords via a browser because Web security was based on protecting URLs, not applications. As applications and databases started being attached to the backend of URLs, it became clear that SSL had limitations.

SSL is CPU intensive and, for the most part, a server can support only a limited number of SSL handshakes. In some cases, SSL accelerators can resolve this performance problem. However, SSL cannot create a user experience, where on a front-end portal, the user puts in one password, and all the other applications on the backend of the portal receive authentication information about that user. A properly implemented Single Sign-On solution will write the front-end authentication through to a central management console on the backend, and share this information between applications for the extent of the user session. Also, SSL only can work between two endpoints, so if a transaction involves three parties such as a customer, a merchant, and a card issuer, you cannot use SSL.

1.1.2 Legacy Single Sign-On

Legacy Single Sign-On products conceptually use the same authentication and authorization architecture structure as Web Single Sign-On, except a portal is not part of the front-end picture. Legacy Single Sign-On enables smooth navigation of the various applications on an intranet through one authentication session. Many legacy Single Sign-On products also offer smart card, PKI, and biometric support. It is worth noting that some industry analysts refer to legacy Single Sign-On as employee Single Sign-On.

1.2 Problem Statement

As the proliferation of applications accelerates, users find themselves losing the authentication war, every application that they are introduced to requires its own authentication or login. Some applications do this because they need to identify the user they are dealing with. Other applications do so because they simply do not trust anyone

else to conduct authentication for them. The evolution of computing has introduced many ways to accomplish authentication, some stronger than others. Single Sign-On represents an attempt to address the “multiple login” issue, as well as other issues that are intrinsically tied to multiple logins.

The importance of Single Sign-On cannot be stressed enough. Many organizations are implementing Single Sign-On because of numerous significant benefits. Those are the benefits of using Single Sign-On in order to overcome the problem arise before implementation of this system.

1.2.1 User productivity

This is the most immediate benefit of implementing Single Sign-On. If we can accomplish in one login what today takes us ten logins to accomplish, one would have to agree that there is inherent efficiency in making this happen. We do no longer have to remember ten accounts and ten passwords. We do not have to remember when we will be asked to change password on application X, what policies govern this password versus the password on application Y, and who do I need to contact when I forget the password on application Z.

1.2.2 Increased security

This is a less tangible but far more important benefit of Single Sign-On. The human brain can remember only so much detail, and eight, nine, or ten different accounts and passwords far exceeds that amount. We all know what users do when they are overloaded. They write their username and passwords down and put them in “safe” places: in desk drawers, under keyboards, or in a file on their computer or PDA. The system needs to manage this for us and it needs to do this securely and efficiently. Some claim that implementing Single Sign-On can actually weaken security. The argument

goes something like this: all we have to do is break one password and we have broken them all. This argument, however, is patently false. If the password used to protect Single Sign-On identity is inherently weak either because it is static or because it has a weak policy engine behind it), then it is probable that other passwords are of equal quality. So breaking any one of them would probably mean that we have now discovered the password far more than half of the other systems at the same time. The time it takes to break a weak password is no time at all. Knowing that the vulnerability of any security system is measured by its weakest link, what is required is to strengthen that link, not throw away the system.

1.2.3 Time and cost savings

Finally, increased productivity and increased security add up to significant time and cost savings for everyone.

1.3 Objectives and Scope of Study

Identity proliferation frustrates employees, compromises network security, drives up administration costs and hampers measures to comply with legislative regulation. Single sign-on (SSO) has long been touted as the panacea to manage the ever growing number of logins that most computer users will face when working with today's modern IT systems. From system server logins to web based internet banking and email programs, users tend to be bombarded with a vast number of different system logins each day with most requiring a different username and password. The concept behind single sign-on is that the user logs in once and the computer then takes over to perform any subsequent logins automatically on behalf of that user. The burden of having to remember many different passwords is thus reduced or even eliminated. The result is user convenience dramatically improves, help desk calls are reduced, and security is strengthened.

Historically, single sign-on has always been viewed as weakening a system's security. This is mainly due to the fact that the authentication which is designed to protect the system is being reduced to a few or even a single credential validation event. The initial login must therefore be quite secure in order to stop any malicious party from gaining unauthorized access. Advancements in secure user login have developed to such a stage that strong authentication can now be assured in a single authentication session. Multifactor authentication is based on the concept of using two or more methods to identify the user. This is generally something the user has (such as smart card or token), something the user knows (a password or PIN), and something that only the user can present (a biometric ID such as from a fingerprint or retina scan). Any good single sign-on implementation should have the flexibility to offer the integration of advanced authentication systems in order to boost the security of the user's login. So, basically the objectives of this Single Sign-On Web Application are to reduce the time spent by system administrator tracking and resetting passwords for various account logins, ease the pain of remembering passwords where user no longer have to remember multiple passwords, and simplify the authentication process for the users and provide advanced security to various applications.

CHAPTER 2

LITERATURE REVIEW AND/OR THEORY

1. “ LDAP’s support by various programming languages offers interesting possibilities. It’s quick to create an extensive address management system for your home intranet with these, if you use languages such as PHP.” [Volker Schwaberow (2001), Open LDAP: Practical Application Organising Principle]

2. “ Clearly, implementing Single Sign-On for any but the smallest and most homogeneous of enterprises is a highly organization-specific task, and certainly no one can provide a true “cook book” for arriving at an agreeable and functional solution for use in all environments. Nevertheless, the authors are convinced that as organizational computing infrastructures become increasingly complex and users become more dissatisfied with the ease of use problems associated with heterogeneous computing environments, system administrators will increasingly be called upon to provide Single Sign-On Solutions for their organizations. [Michael Fleming Grubb and Rob Carter (1998), Single Sign-On and the System Administrator]

3. “PHP is a powerful, flexible open source scripting language typically used to generate dynamic content in web pages. (PHP is similar to Perl, but much less complicated. For more background, visit OTN's [Open Source Developer Center](#).) Out of the box, PHP provides a rich set of features and services including LDAP, IMAP, SNMP, NNTP, POP3, HTTP, XML, XSL, and database access modules for most database products including Oracle. PHP can run as a CGI under Apache or be

configured as an Apache Web Server module.” [David Jason Bennett (2004), Single Sign-On for PHP Pages]

4. “A business objective for many institutions is to construct a capability that enables users to access all of their relationships with the institution through a single portal website that requires only a single login, regardless of the separate applications that may invoke. Typically, each application is constructed to require its own login, often with unique login ID and password. The burden is usually placed on the user to remember and enter these passwords for each function or application they invoke. A single login portal would remove that burden from the user and facilitate their access to those applications that are authorized to them. This single login capability could also lead to providing the institution with a unified view of each user’s interactions, improving the quality of the “real-time” service. “ [Hewlett Packard Development Company (2003), Single Sign On Capability]

CHAPTER 3

METHODOLOGY/PROJECT WORK

For this Single Sign-On system, the consideration has being made to use PHP language as well as Apache server. On this subtopic, we will discuss about PHP and Apache in terms of how their works, their advantages, their mechanisms and many else. In order to make sure that PHP and Apache are “well-communicate”, we have to download appserv-win32-2.4.1, a kind of software to achieve those relations. We also will use MYSQL as the database part. We also will discuss about how the Single Sign-On mechanisms and what tools in terms of equipment, software and hardware used to achieve the objective.

3.1 PHP

According to the official PHP Web site, PHP is “a server-side, cross-platform, and HTML embedded scripting language.” This may sound like a mouthful, but it’s fairly straightforward and meaningful when broken down into its parts. First, server-side refers to the fact that everything PHP does occurs on the server (as opposed to on the client, which is the Web site viewer’s computer). A server is simply a special computer that houses the pages that you see when you go to a Web address with your browser.

PHP is cross-platform, meaning that it can be used on machines running almost any operating system--Unix, Windows NT, Macintosh, OS/2, to name the most popular ones. Again, that’s the server’s operating system, not the client’s. Not only that, but unlike with most other programming languages, your work can be switched from one platform to another with very few or no modifications. To say that PHP is HTML embedded means you can put it into your HTML code--HTML being the code with which all Web pages are built. Therefore, scripting with PHP can be only slightly more complicated than hand-coding HTML. If you can make a basic HTML Web page, you can make a dynamic one, too.

Last, PHP is a scripting language, as opposed to a programming language. This means that PHP is designed to do something only after an event occurs; for example, when a user submits a form or goes to a URL. Programming languages such as Java, C, and Perl can write standalone applications, while PHP cannot. The most popular Web scripting language is JavaScript, which commonly handles events that occur within the browser (for example, when a mouse goes over an image), and is not dissimilar to PHP, although JavaScript is a client-side technology. In fact, if you can already work with JavaScript, you won't have a problem mastering PHP.

3.1.1 Why PHP?

Webmasters learned a long time ago that HTML alone cannot produce enticing and lasting Web sites. That's why server-side technologies such as CGI scripts have gained widespread popularity. With them, Web page designers can create dynamically generated Web applications. Often database-driven, these advanced sites require less manual work to update and maintain than static HTML pages, and they allow for e-commerce and other advanced transactions. PHP lets you exponentially expand what you can do with the World Wide Web. The advantage PHP has over basic HTML is that the latter is limited and one-sided. Visitors to HTML-only sites see simple pages that aren't customized for them and don't display any dynamic behavior. With PHP, you can create exciting and original documents based on many factors--the time of day or whether the user is a repeat visitor, for example. PHP can also interact with databases and files, handle email, and do many other things that HTML cannot.

But the question remains: Why should a Web designer use PHP to make a dynamic Web site instead of CGI (Common Gateway Interface), ASP (Active Server Pages), or JSP (Java Server Pages)? First, PHP is both faster to program in and faster to execute than CGI scripts. It won't get into too much detail, but suffice it to say that compared to full programming languages like Java, C, or Perl, PHP is much easier to learn and use. People without any formal programming training can write PHP scripts with ease after

reading a book. In comparison, ASP requires an understanding of VBScript, and CGI requires Perl (or C), both of which are more difficult to learn, not to mention presumably more expensive to develop in.

Second, PHP was written specifically for dynamic Web-page creation, whereas Perl (and VBScript and Java) were not, inferring that, by its very intent, PHP can perform certain tasks faster and easier than its alternatives. The final argument for learning PHP is that, once you do, and as its popularity continues to grow (it is already being used on nearly six million Web sites), you will find yourself well ahead of the learning curve on this, the latest “next big thing” in the world of Internet technology.

3.1.2 How PHP works

When you go to a website (here we used <http://www.DMCinsights.com/php/index.php> as an example here), your Internet Service Provider directs your request to the server that holds the <http://www.DMCinsights.com/php/index.php> information. Because this site was designed in PHP, the server reads the PHP and processes it according to its scripted directions. In this example, the PHP code tells the server to send the appropriate Web page data to your browser. This data is in the form of HTML that the browser can display as it would a standard HTML page. In short, PHP creates an HTML page on the fly based on parameters of my choosing; the server contains no static HTML pages.

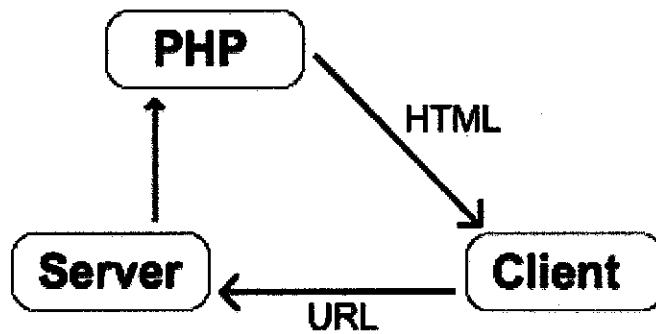


Figure 3.1 – Relationship between Server and Client through PHP

This graphic demonstrates how the process works between a Client, the Server, and a PHP module (an application added to the server to increase its functionality) to send HTML back to the browser (albeit in very simplistic terms). All server-side technologies (ASP, for example) use some sort of third-party module on the server to process the data that gets sent back to the client. With a purely HTML-generated site, the server merely sends the HTML data to the Web browser; there is no server-side interpretation.



Figure 3.2 – Relationship between Server and Client

Compare this direct relationship of how a server works with basic HTML to that of **Figure 1.1**. This is also why HTML pages can be viewed in your browser from your own computer since they do not need to be "served," but dynamically generated pages need to be accessed through a server which handles the processing.

To the end user and their browser, there may not be an obvious difference between what <http://www.DMCinsights.com/php/index.php> and <http://www.DMCinsights.com/php/index.html> look like, but how the pages arrived at that point are critically different. The major difference: By using PHP, you can have the server dynamically generate the HTML code. In this example, the index.php page referenced above displays news items that it retrieves chronologically from a database. Dynamic Web page creation is what sets apart the less appealing, static sites from the more interesting, and therefore more visited, interactive ones.

3.2 Apache Server

Apache is the most popular Web server software on the Internet. The true secret of Apache's success is that the source code is freely available. This means that anyone who wants to add features to their Web server can start with the Apache code and build on it. Indeed, some of Apache's most important modules began as externally developed projects. To encourage this kind of external development, all binary distributions now come with a complete copy of the source code that's ready to build. Examining the source code can be instructive and educational, and sometimes, it can even turn up a bug—such is the power of open peer review. When a bug is found in Apache, anyone can post a fix for it to the Internet and notify the Apache development team. This produces rapid development of the server and third-party modules, as well as faster fixes for any bugs discovered. It's also a core reason for its reputation as a secure Web server.

3.2.1 How Apache works

Apache doesn't run like a user application such as a word processor. Instead, it runs behind the scenes, providing services for other applications that communicate with it, such as a Web browser. In Unix terminology, applications that provide services rather than directly communicate with users are called daemons. Apache runs on Windows NT, where the same concept is known as a service. Windows 95/98 and Windows ME

aren't capable of running Apache as a service; it must be run from the command line (the MS-DOS prompt or the Start menu's Run command), even though Apache doesn't interact with the user once it's running.

Apache is designed to work over a network, so Apache and the applications that talk to it don't have to be on the same computer. These applications are generically known as *clients*. Of course, a network can be defined as anything from a local intranet to the whole Internet, depending on the server's purpose and target audience. The most common kind of client is of course a Web browser; most of the time when we talk about *client*, it means *browser*. However, there are several important clients that aren't browsers. The most important are Web robots and crawlers that index Web sites, but don't forget streaming media players, news ticker applications, and other desktop tools that query Internet servers for information. Web proxies are also a kind of client because they forward requests for other clients.

The main task of a Web server is to translate a request into a response suitable for the circumstances at the time. When the client opens communication with Apache, it sends Apache a request for a resource. Apache either provides that resource or provides an alternative response to explain why the request couldn't be fulfilled. In many cases, the resource is a Hypertext Markup Language (HTML) Web page residing on a local disk, but this is only the simplest option. It can be many other things, too—an image file, the result of a script that generates HTML output, a Java applet that's downloaded and run by the client, and so on. Apache uses HTTP to talk with clients. It's a request/response protocol, which means that it defines how clients make requests and how servers respond to them: Every HTTP communication starts with a request and ends with a response. The Apache executable takes its name from the protocol, and on Unix systems is generally called `httpd`, short for *HTTP daemon*.

3.2.2 Running Apache : UNIX vs Windows

Apache was originally written to run on Unix servers, and today it's most commonly found on Linux, BSD derivatives, Solaris, and other Unix platforms. Since Apache was ported to Windows 95 and NT, it has made substantial inroads against the established servers from Microsoft and other commercial vendors—a remarkable achievement given the marketing power of those companies in the traditionally proprietary world of Windows applications. Because of its Unix origins, Apache 1.3 was never quite as good on Windows as it was on Unix, but with Apache 2, programmers have completely redesigned the core of the Apache server. One major change is the abstraction of platform-specific implementation details into the Apache Portable Runtime (APR), and the server's core processing logic has been moved into a separate module, known as a Multi Processing Module (MPM). As a result, Apache runs faster and more reliably on Windows because of an MPM dedicated to those platforms. NetWare, BeOS, and OS/2 also benefit from an MPM tuned to their platform-specific needs.

Apache runs differently on Unix systems than on Windows. When we start Apache 1.3 on Unix, it creates (or forks) several new child processes to handle Web server requests. Each new process created this way is a complete copy of the original Apache process. Apache 2 provides this behavior in the prefork MPM, which is designed to provide Apache 1.3 compatibility. Windows doesn't have anything resembling the fork system call, so Apache was extensively rewritten to use the native Windows threads. Theoretically, this is a much more efficient and lightweight solution because threads can share resources (thereby reducing their memory requirements). It also allows more intelligent switching between tasks by the operating system. However, Apache 1.3 used the Windows POSIX emulation layer (a Unix compatibility standard) to implement threads, which meant that it never ran as well as it theoretically would have. Apache 2 uses native Windows threads directly, courtesy of the APR, and accordingly runs much more smoothly.

Thread support in Apache 2 for the Unix platform is found in the worker, leader, threadpool, and perchild MPMs, which provide different processing models depending on your requirements. The new architecture coupled with the benefits of threaded programming provide a welcome boost in performance and also reduce the differences between Windows and Unix, thus simplifying future development work on both platforms. Apache is more stable on Windows NT, 2000, and XP than on Windows 9x and ME because the implementation of threads is cleaner on the former. To run Apache on Windows with any degree of reliability, choose an NT-derived platform because it allows Apache to run as a system service.

3.2.3 Apache configuration

Apache is set up through configuration files in which directives can be written to control Apache's behavior. Apache supports an impressive number of directives, and each module that's added to the server provides more. The approach Apache takes to configuration makes it extremely versatile and gives the administrator comprehensive control over the features and security provided by the server. It gives Apache a major edge over its commercial rivals, which don't offer nearly the same degree of flexibility and extensibility. It's also one of the reasons for Apache's slightly steeper learning curve, but the effort is well worth the reward of almost complete control over every aspect of the Web server's operation.

The drawback to Apache's versatility is that, unlike other commercial offerings, there's currently no complete solution for configuring Apache with a Graphical User Interface (GUI) editor—for involved tasks, we must edit the configuration by hand. That said, there are some credible attempts at creating a respectable configuration tool to work with Apache's power and flexibility. The drawback is that many configuration tools handle only the most common configuration tasks, so the more advanced your needs become, the more we'll find yourself editing the configuration directly. The fact that

we're editing in a GUI editor's window doesn't alter the fact that the GUI can help we only so much.

One of Apache's greatest strengths is its modular structure. The main Apache executable contains only a core set of features. Everything else is provided by modules (as in Figure 1.3), which can either be built into Apache or be loaded dynamically when Apache is run. Apache 2 takes this concept even further, removing platform-specific functionality to MPMs and subdividing monolithic modules such as `mod_proxy` and `mod_cache` into core and specific implementation submodules. This allows us to pick and choose precisely the functionality we want. It also provides an extensible architecture for new proxy and cache types. Consequently, the Web server administrator can choose which modules to include and exclude when building Apache from the source code, and unwanted functionality can be removed. That makes the server smaller, require less memory, and less prone to misconfiguration. Therefore, the server is that much more secure. Conversely, modules not normally included in Apache can be added and enabled to provide extra functionality.

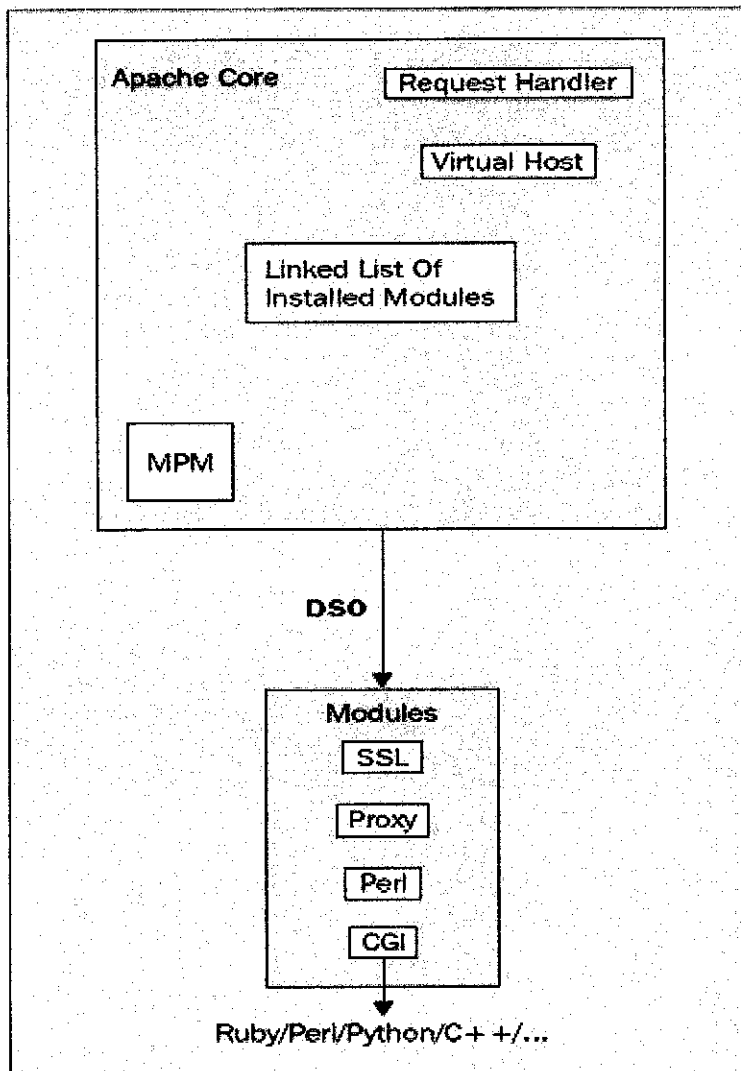


Figure 3.3 – Apache modules

Apache also allows modules to be added so we don't have to rebuild Apache each time you want to add new functionality. Adding a new module involves simply installing it and then restarting the running Apache server—nothing else is necessary. To support added modules, Apache consumes a little more memory than otherwise, and the server starts more slowly because it has to load modules from disk. This is a minor downside but possibly an important one when high performance is a requirement. Additionally, the supplied `apxs` tool enables you to compile and add new modules from the source code to your server using the same settings that were used to build Apache itself. It's

this flexibility, Apache's stability and performance, and the availability of its source code that makes it the most popular Web server software on the Internet.

3.3 MYSQL

Pronounced "my ess cue el" (each letter separately) and not "my SEE kwill." MySQL is an open source RDBMS that relies on SQL for processing the data in the database. MySQL provides APIs for the languages C, C++, Eiffel, Java, Perl, PHP and Python. In addition, OLE DB and ODBC providers exist for MySQL data connection in the Microsoft environment. A MySQL .NET Native Provider is also available, which allows native MySQL to .NET access without the need for OLE DB. MySQL is most commonly used for Web applications and for embedded applications and has become a popular alternative to proprietary database systems because of its speed and reliability. MySQL can run on UNIX, Windows and Mac OS. MySQL is developed, supported and marketed by MySQL AB. The database is available for free under the terms of the GNU General Public License (GPL) or for a fee to those who do not wish to be bound by the terms of the GPL.

3.3.1 MYSQL Web Application

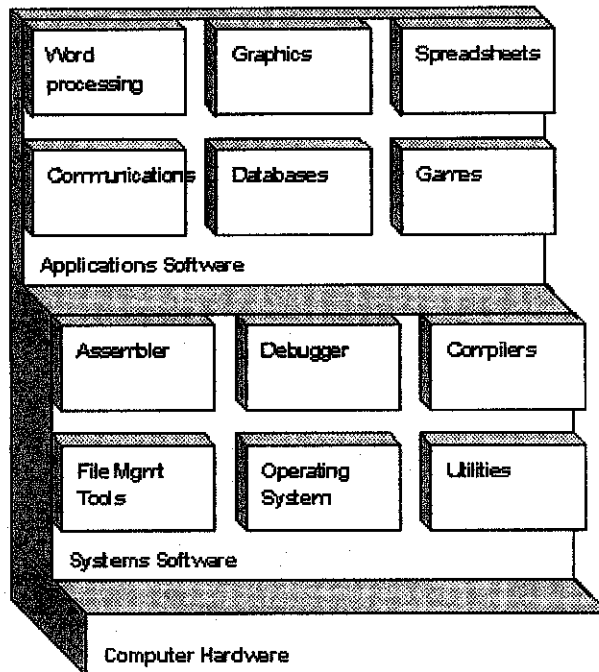


Figure 3.4 – Model of Web Application

A program or group of programs designed for end users. Software can be divided into two general classes: *systems software* and *applications software*. Systems software consists of low-level programs that interact with the computer at a very basic level. This includes operating systems, compilers, and utilities for managing computer resources. In contrast, applications software (also called *end-user programs*) includes database programs, word processors, and spreadsheets. Figuratively speaking, applications software sits on top of systems software because it is unable to run without the operating system and system utilities.

3.3.2 Why Use MYSQL?

MySQL is very fast, reliable, and easy to use. MySQL also has a very practical set of features developed in close cooperation with its users. It is also Open Source and therefore freely accessible. MySQL is used to access databases on the internet due to its connectivity, speed and security. It was originally developed to manage large databases at a much faster speed than the solutions that previously existed. MySQL has for several years, been thriving in the challenging areas of production.

3.3.3 MYSQL Database Setup

Below here is the steps taken to setup the MYSQL database. This is to show the sequence of the process before the whole system can be used.

Step 1

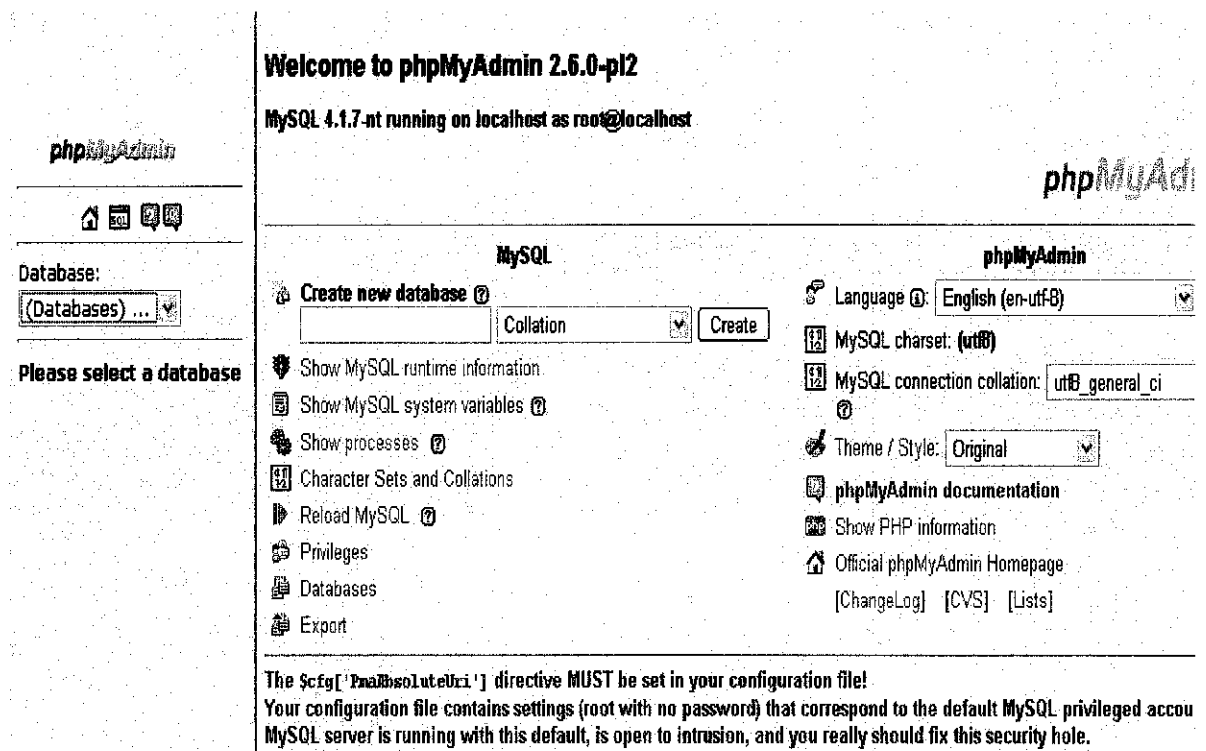


Figure 3.5 – MYSQL Control Panel

First of all, we need to open the MYSQL Control Panel through Internet Explorer using dynamic IP Address of <http://127.0.0.1/db/>. This kind of page will be appearing like Figure 3.5.

Step 2

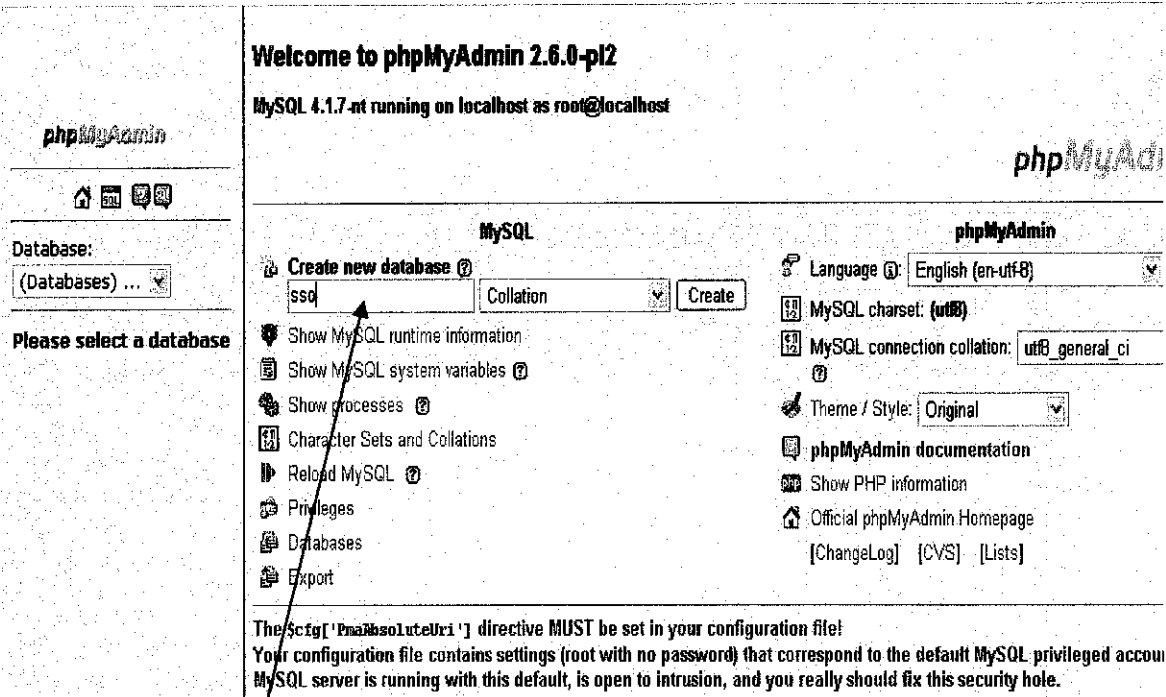


Figure 3.6 – Create database called “sso”

To create database named “sso”, fill in this blank like in the figure.

In this step, we want to create the database called “sso”. Developer have to enter the required name for the database (in this example, the name is “sso”) into the create textbox like in the Figure 3.6.

Step 3

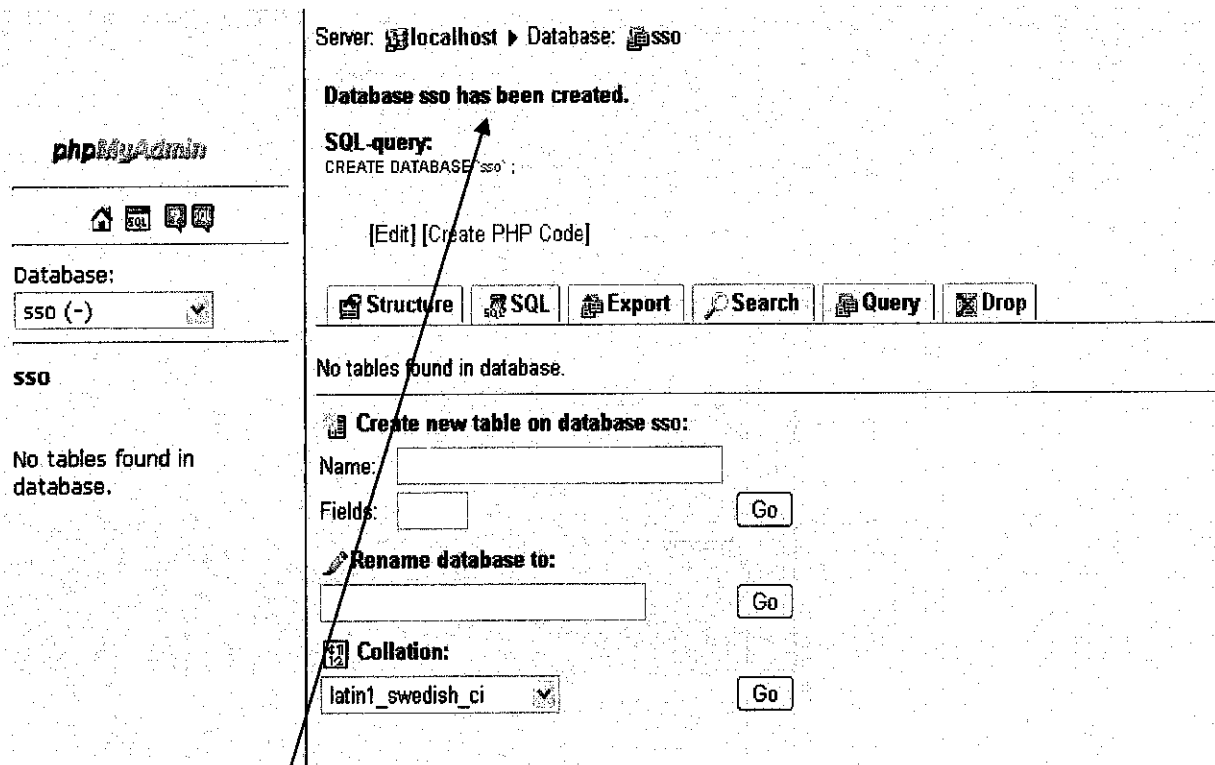


Figure 3.7 – Database “sso” has been created

When the database “sso” have being created, this type of message will appear, “Database sso has been created.”

When the required database we want to create (as in the example, we create database called “sso”) is successfully created, the message will appear as shown in Figure 3.7.

Step 4

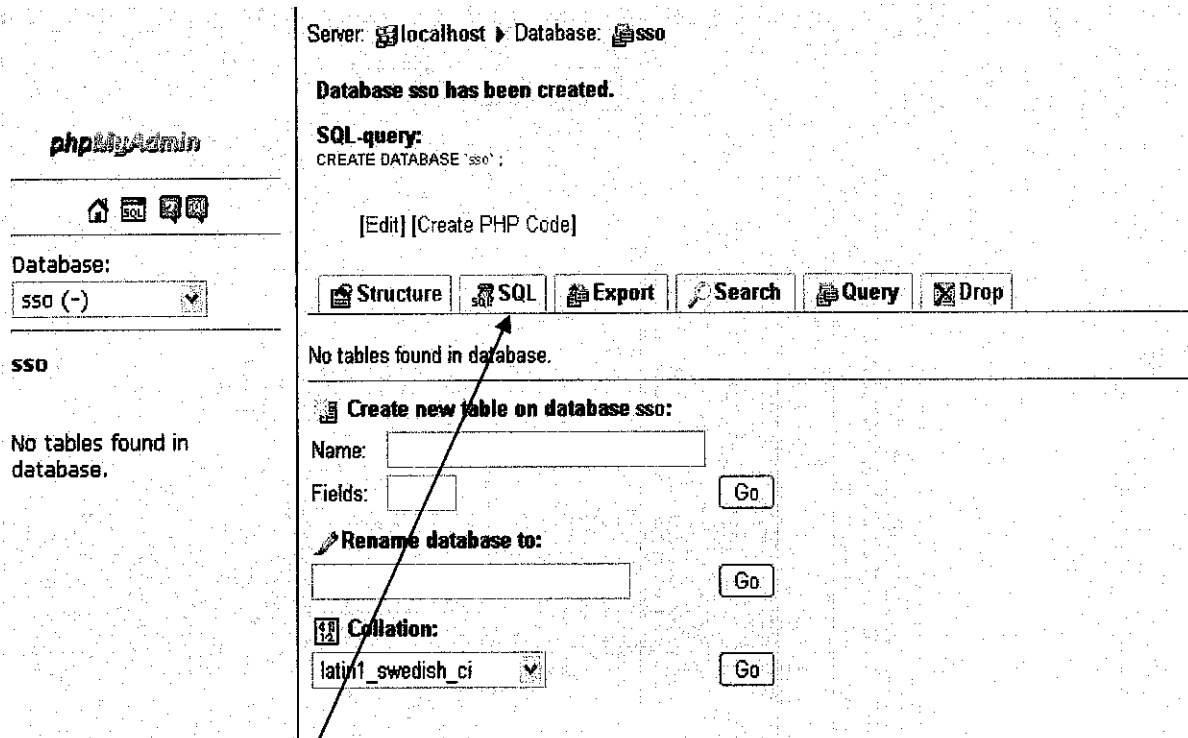


Figure 3.8 – SQL button for entering data in the database.

In order to enter the data in the previous created database, we need to click on the “SQL” button.

Developer need to click on the “SQL” button as in Figure 3.8 in order to enter data into the empty database. This is useful to enter user’s username and password so they have grant to enter into the particular application.

Step 5

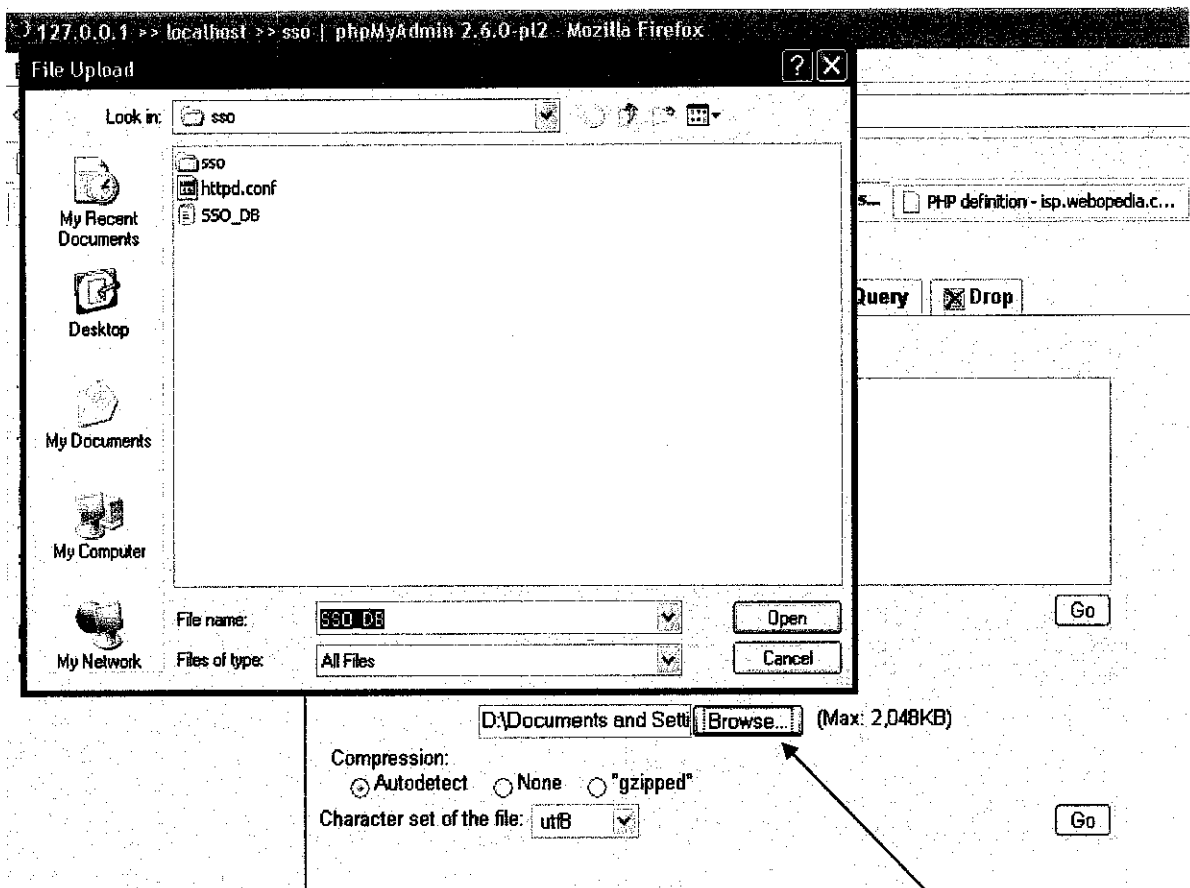


Figure 3.9 – Upload the database data entry

Click on the “Browse” button as shown in Figure 3.9 to upload the database data. The “File Upload” box will appear and developer just have to choose the file path, and then click “Open” button. The file path will be appear in the “Browse” textbox.

Step 6

Server: localhost Database: sso

Structure SQL Export Search Query Drop

Run SQL query/queries on database sso: ?

Database: sso (-)

sso

No tables found in database.

☒ Show this query here again Go

Or

Location of the textfile: D:\Documents and Settings\Browse... (Max: 2,048KB) Browse...

Compression: ☒ Autodetect ☐ None ☐ "gzipped"

Character set of the file: utf8 Go

Figure 3.10 – Upload the location of the textfile

Click “Go” button to make sure that database entry is success.

When the developer seen the path name in the textfield, click “Go” as shown in the Figure 3.10.

Step 7

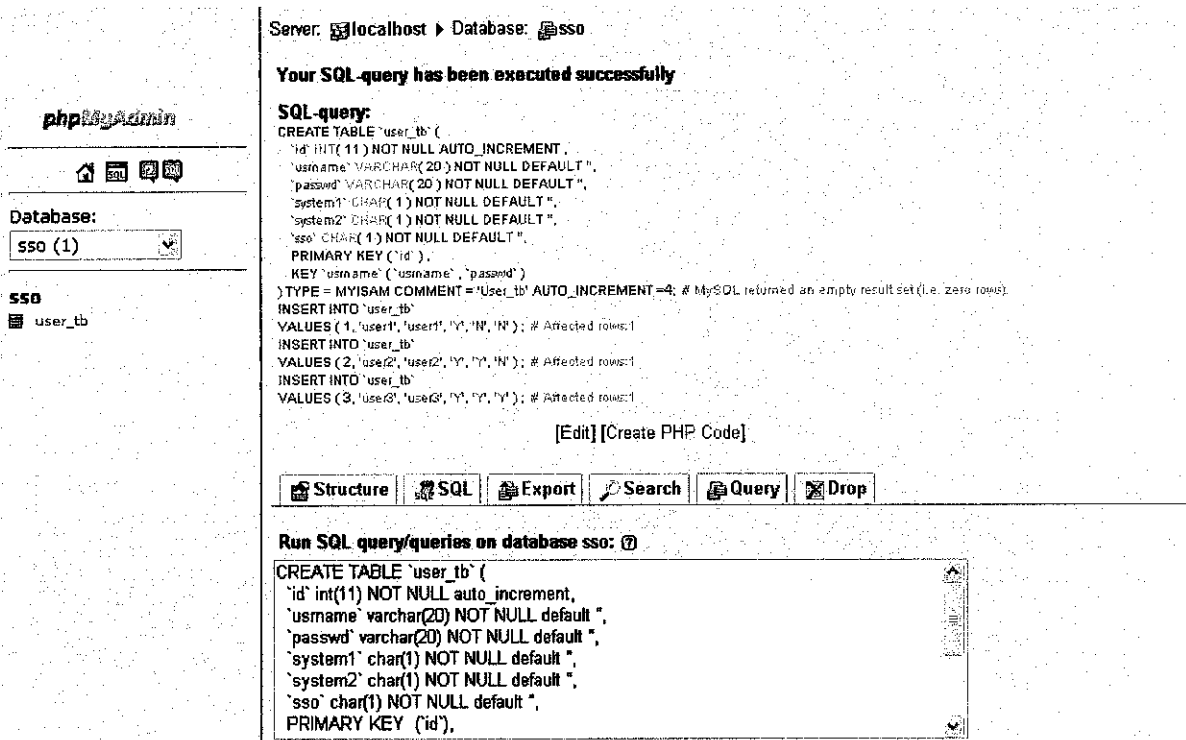


Figure 3.11 – MYSQL database successfully upload

When all the steps have been followed, the successfully database data will be appeared as shown in Figure 3.11. The data in the database can be edit, and if the developer wants to delete this database, just click on the “Drop” button. All this only can be done by the administrator.

3.4 Single Sign-On Mechanisms

It is time to look at one of Single Sign-On architectures, which is scripting, where it serves its strengths as well as its weakness. This is by far, the simplest approach of any. It is fairly easy to implement, because it is noninvasive to either client or server applications. Its primary goal is to automate the login procedure. At the same time, it is not concerned with securing any of the existing applications. Also, scripts are not necessarily intuitive to write and certainly present a scalability concern if they need to be managed on each and every workstation.

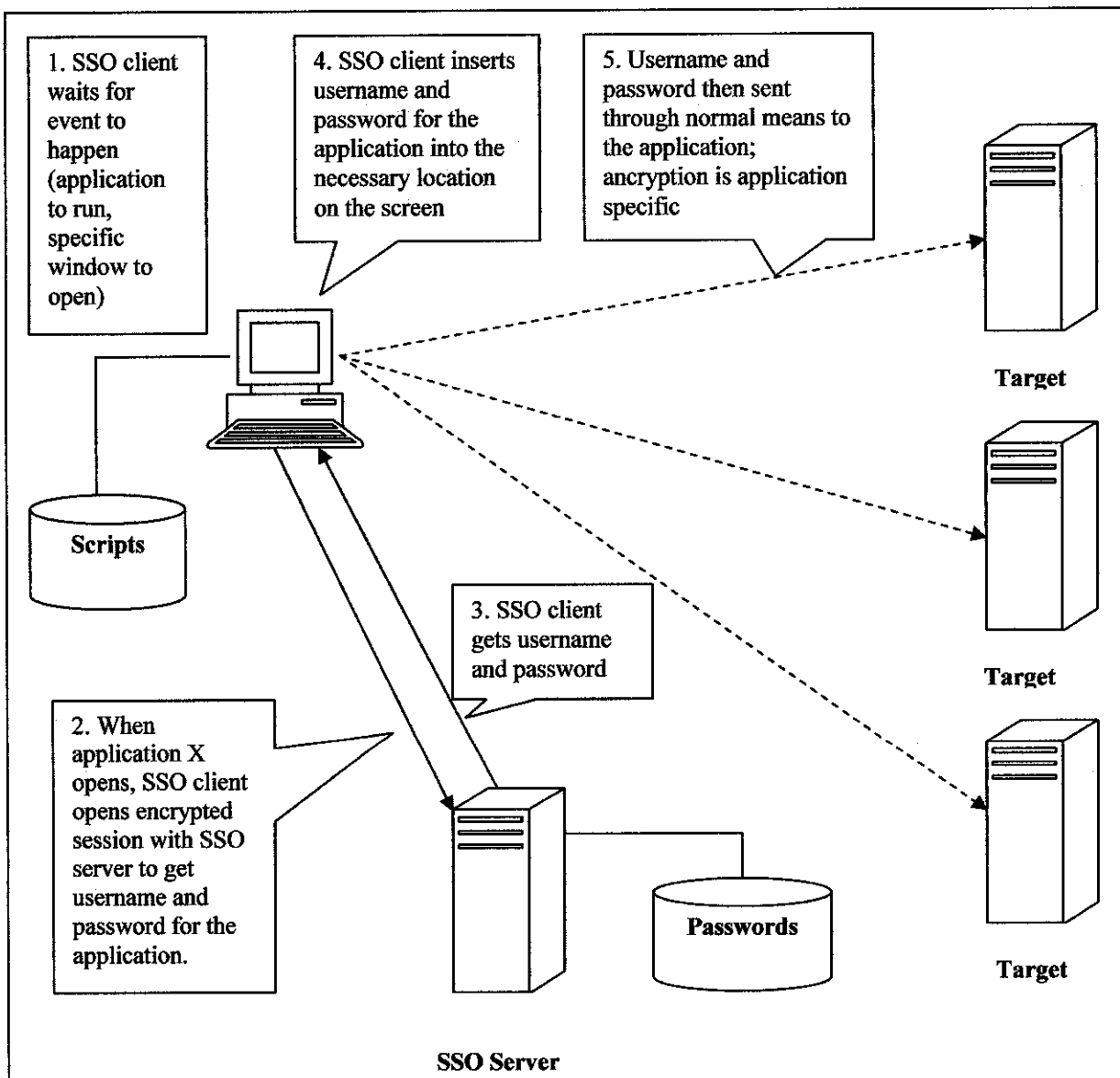


Figure 3.12 – An Single Sign-On Architecture Utilizing Scripting

3.5 Tools required

Those are the list of software and hardware that have being used in order to finish the project.

3.5.1 Appserv-win32-2.4.1

At this time AppServ has been split to version 2.4.x and 2.5.x. 2.4.x is a Superb stable version work for all user, by the way this version use PHP 4.x because work fine with your old PHP code.

3.5.2 Microsoft Windows XP

3.5.3 DELL Personal Computer

3.5.4 Pentium® 4 CPU 2.26GHz

3.5.5 256 MB of RAM

CHAPTER 4

RESULTS AND DISCUSSION

Below this are the screenshots for the Single Sign-On system. Each of the screenshots, there are description tell about the flow of the system.

Screenshot 1

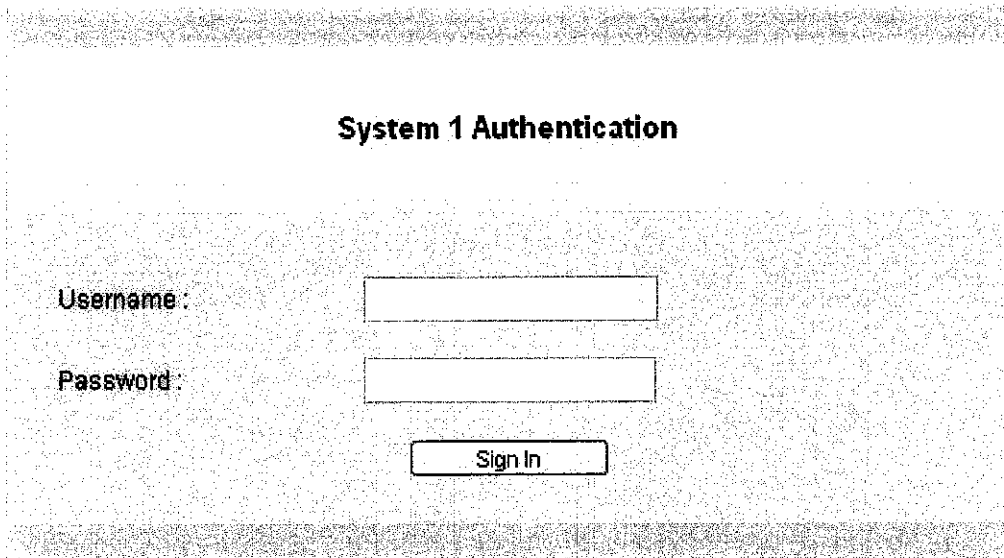
A screenshot of a web application login page titled "System 1 Authentication". The page has a light gray background with a darker gray header and footer. In the center, there is a white rectangular area containing the login form. The form consists of two text input fields: the first is labeled "Username:" and the second is labeled "Password:". Below these fields is a "Sign In" button. The text "System 1 Authentication" is centered above the input fields.

Figure 4.1 – System 1 Web Application Login Page

This is the System 1 Web Application Login Page where users who want to enter the application of System 1, they need to have the account consist of their own username and password. If the users don't have the account, they need to request it with the system administration, and then they have grants to enter into the System 1 application.

Screenshot 2

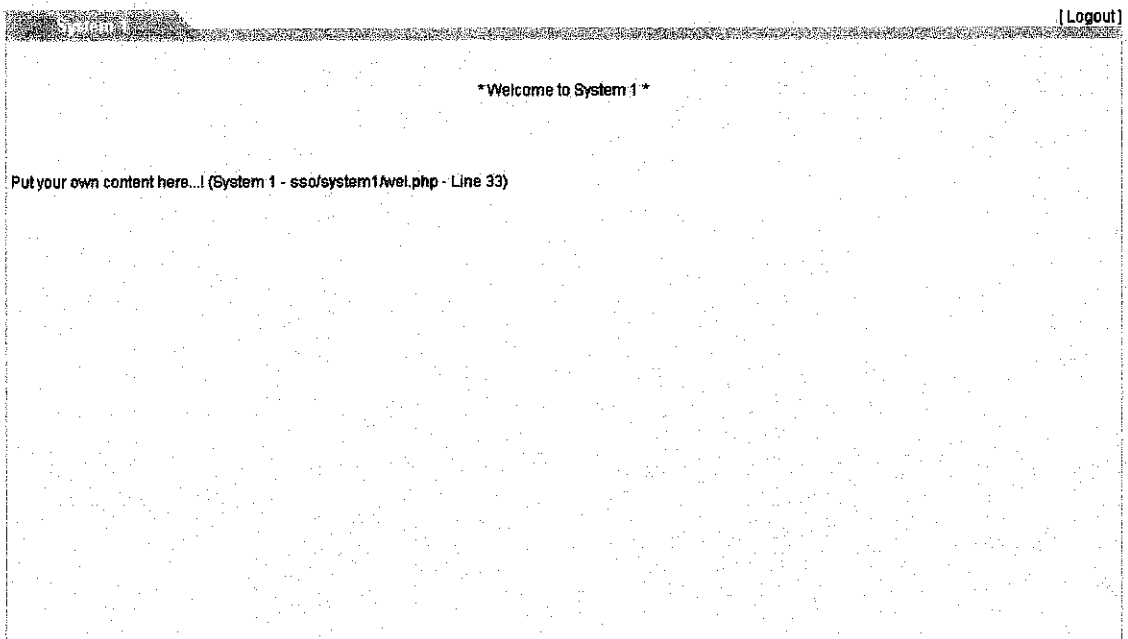


Figure 4.2 – System 1 Web Application Main Page

After the users have successfully entered their username and password, they will see this page showing the content of System 1 (in this figure, it only a blank content because it just for a purpose of testing). Here, it also provided the “Logout” button to make the users’ task easy in case they want to enter another account or just want to logout.

Screenshot 3

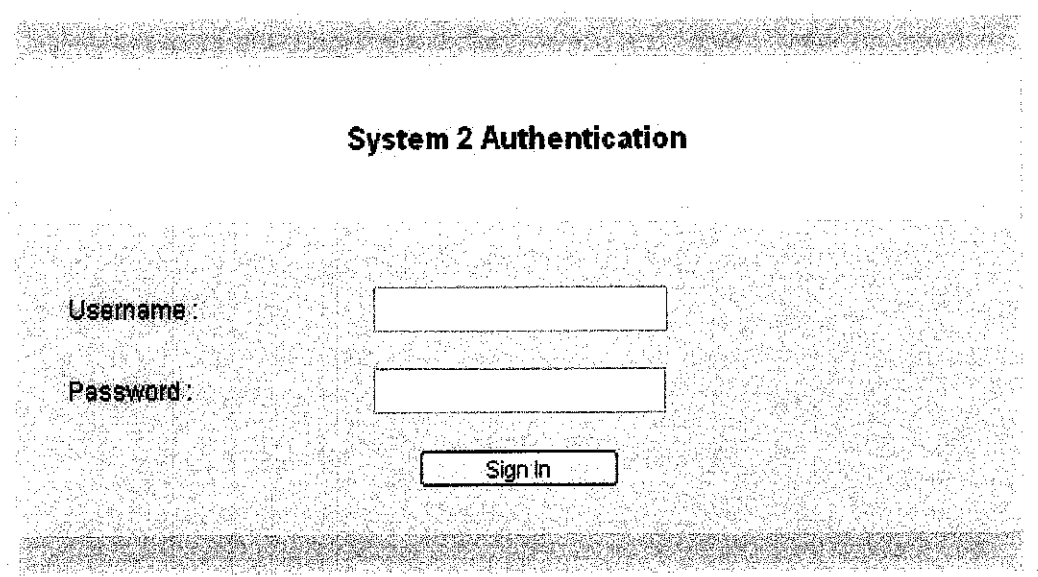


Figure 4.3 – System 2 Web Application Login Page

This is the page showing the System 2 Web Application Login Page. As System 1, users need to have their own account in order to enter into this application. When the wrong username and password are entered, the system will not give an access to that user. The warning message will appear inform the user that they have entered the wrong username and/or password.

Screenshot 4

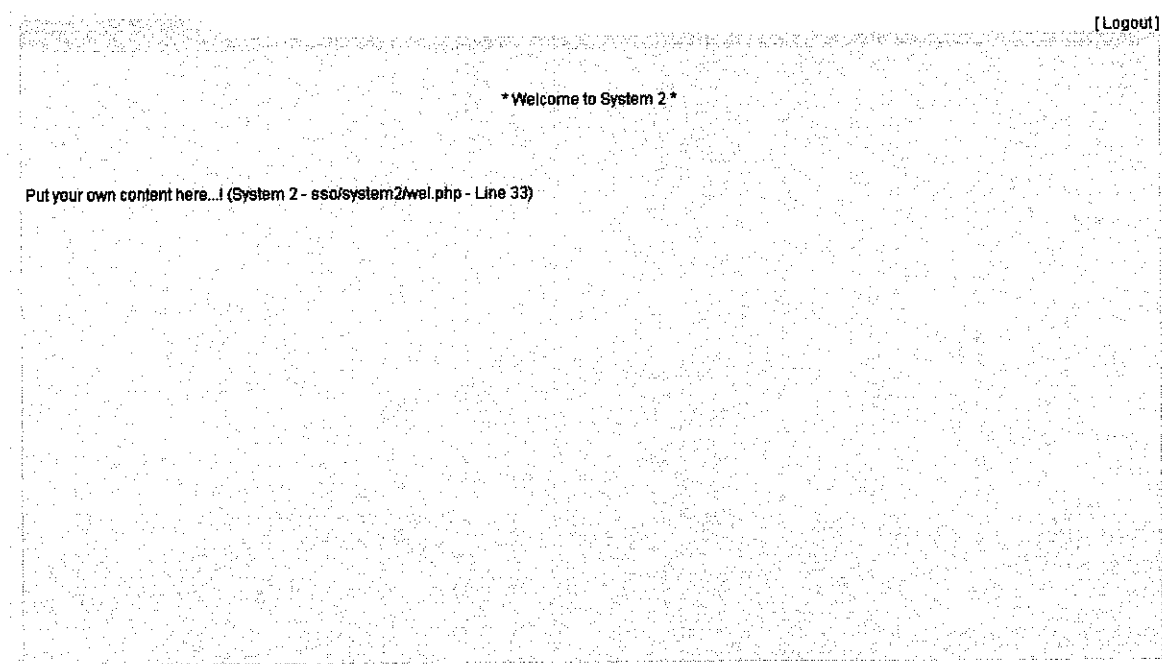


Figure 4.4 – System 2 Web Application Main Page

This page also same along with the System 1 Main Page after the user successfully entered their username and password. As what the content shows, it also have the logout button to ease user to logout from this system. The content of this page also have being limited as this only for a testing period of time, showing that this system is working.

Screenshot 5

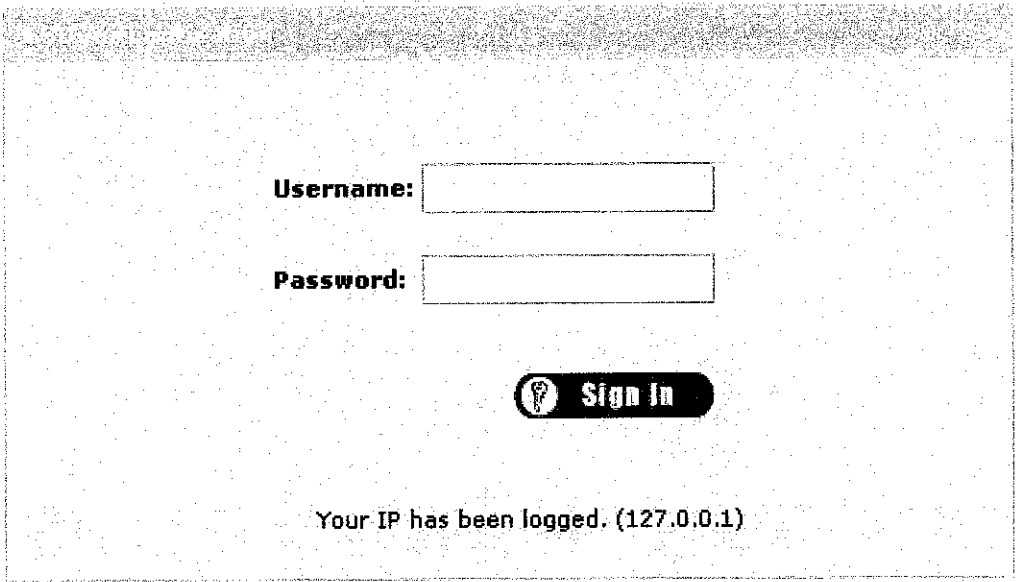


Figure 4.5 – Single Sign-On Web Application Login Page

This is the main Web application of this system, the Single Sign-On Web Application Login Page. This is also need the user account to enter into this application. Below that, it says that the user IP has been logged. This is because this system is using the dynamic IP, so once the user login into this page using their own IP address, their IP address will be logged to IP address, 127.0.0.1.

Screenshot 6

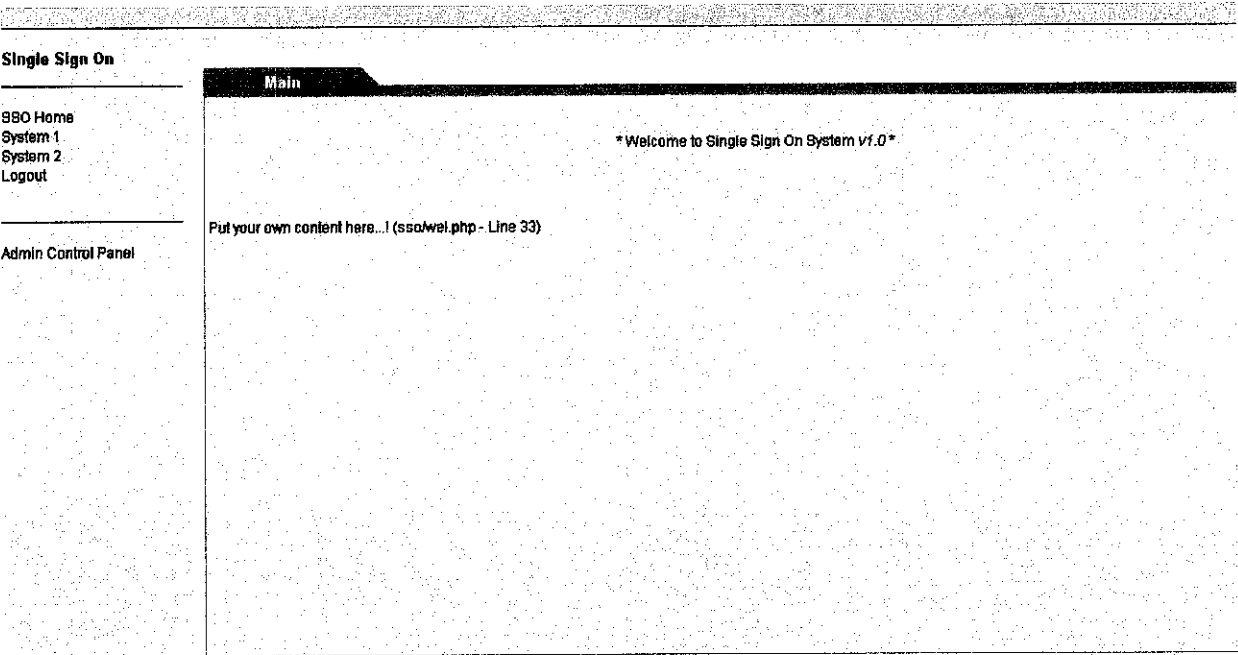


Figure 4.6 – Single Sign-On Web Application Main Page

This page showed the usage of Single Sign-On Web Application. Here, it shows that when user have being given authentication to access to System 1, System 2 and Single Sign-On Web Application, at the left side of the application. It means when the user want to go to the System 1 application, he/she just have to click at the required link and the System 1 Main Page will appeared at the right side of this system. It also has the “Logout” button to ease user to logout from this system. Below the “Logout” button, it showed the Admin Control Panel link. That is for the usage of system administration where they can change whose can or cannot enter in either one of the web applications.

Screenshot 7

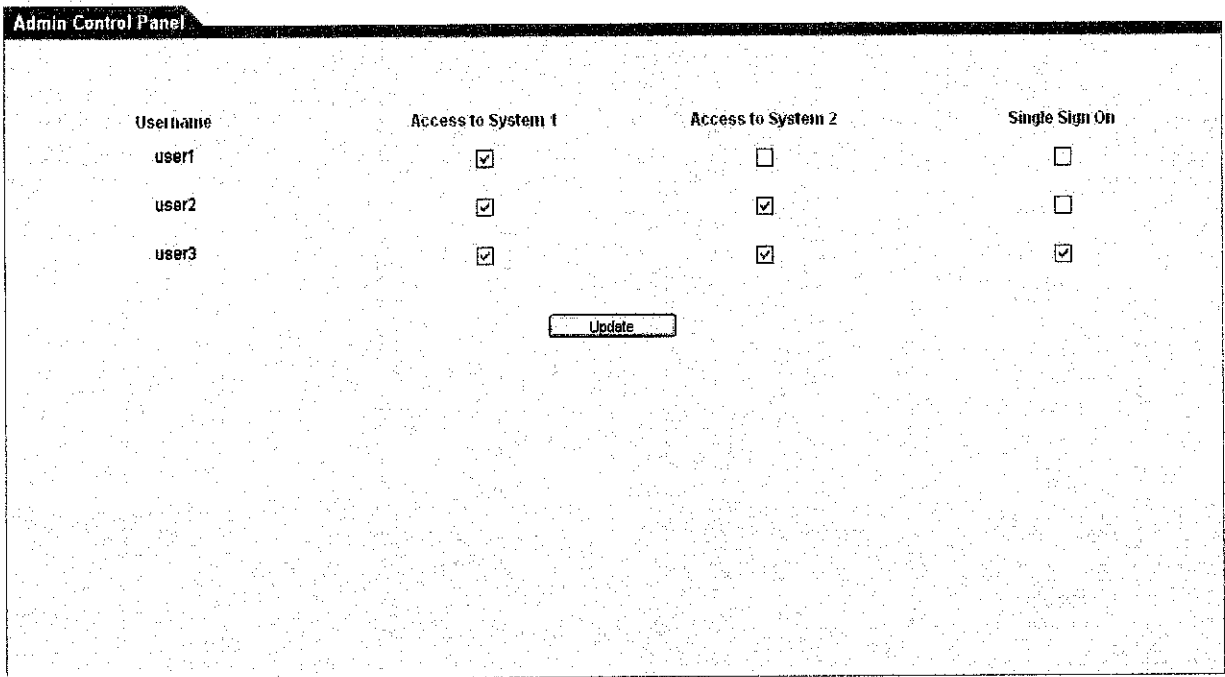


Figure 4.7 – Admin Control Panel Page

Here, it showed that the system administration can change users’ grants for each of the systems. For example here, for user1, he/she can only just to have access to System 1. For user2, he/she have access to enter for System 1 and System 2. But for user3, he/she have being given the full access for all the systems, System 1, System 2 and Single Sign-On system. After system administration has set all the setting for particular user, he/she click on the “Update” button. Then the previous page will be appeared.

Based on these results, any enhancement can be made on this system. Any involvement of third party for authentication also can be included as enhancement. For example, if user wants to have a direct authentication to Yahoo Mail account through Single Sign-On System, it can be made by altering the code for this system.

CHAPTER 5

CONCLUSION AND RECOMMENDATION

Basically, Single sign-on (SSO) is mechanism whereby a single action of user authentication and authorization can permit a user to access all computers and systems where he has access permission, without the need to enter multiple passwords. Single sign-on reduces human error, a major component of systems failure and is therefore highly desirable but difficult to implement. Web single sign-on (Web-SSO), also called Web access management (Web-AM) works strictly with applications and resources accessed with a web browser. Access to web resources is intercepted, either using a web proxy server or by installing a component on each targeted web server. Unauthenticated users who attempt to access a resource are diverted to an authentication service, and returned only after a successful sign-on. Cookies are most often used to track user authentication state, and the Web-SSO infrastructure extracts user identification information from these cookies, passing it into each web resource.

Below this is a list showing the advantages and disadvantages of the chosen Single Sign-On Architectures. Based on this table, some enhancements can be made in order to reduce the number of disadvantages occur in term of the development of Single Sign-On system.

Architecture	Advantages	Disadvantages
Scripting	4 Simple 5 Easy to use, easy to adapt 6 Learn intrusive to application	7 No encryption 8 Distribution and management issues if kept on desktop 9 Security risk if password is stored in the script 10 Unable to provide multitier authentication

Table 5.1 – Advantages and Disadvantages of Single Sign-On Architectures

In the “push” scenario, an HTML form is used on the Source website to include SAML assertions in a hidden field, and these assertions are “pushed” to the Destination website as a part of an HTTP Post.

REFERENCES

1. Caleb Racey, Web based Single Sign on, NCL Publishing Company.
2. *Mahesh Bhaska, Noor-E-Saba Hakim, Sam Lam – Group Panama, Secure Single Sign On, Harvard University Extension School.*
3. *Davis Marasco, Patrick Botz, WebFacing and Single Sign-On :Exploiting Identity Tokens in Multi-tier Web Applications*
4. *David P. Kormann and Aviel D. Rubin, Risks of the Passport Single Sign-On Protocol, AT&T Labs*
5. *Joel Farrell. Peter Greene (2004), Single Sign-On Guidelines, MedBiquitous Technical Steering Committee.*
6. *Alan Lawson (2002), Who to sign-up with for a single sign-on standard, Butler Group Review Journal Article.*
7. *Toni Nykänen (2002), Secure Cross-Paltform Single Sign-On Solution for the World-Wide Web, Department of Computer Science and Engineering Helsinki University of Technology.*