mytab

SYSTEM AND PROGRAM DESIGN

Table of Contents

1. General Overview

1.1 Purpose

MyTab is a web application which acts as a single safe platform to hold multiple accounts of a user and enables accessing these accounts with just one click. The application thus frees the users from having to remember the passwords of every account. It also saves valuable user time by avoiding the need for users to enter their credentials manually each time.

1.2. Scope & Requirements Overview

The aim of this project is to create a Phase 1 prototype web site for a start-up company. The prototype will be used for demonstration to potential investors and as a prototype of the user interface for the eventual production system.

The goal of MyTab.com is to provide web users with a "master account," through which they can access and navigate their many online accounts through a single, safe platform – eliminating the need to continually retype usernames and passwords as well as open new tabs inside their web browser to access accounts.

The initial scope of the project included supporting six different social media sites like Facebook, Gmail, LinkedIn, Twitter, etc. After a thorough technical feasibility analysis it has been concluded that it is technically infeasible to access these sites within a single tab as these websites enforce "same origin policy" which forbids the use of framing.

Taking these findings into account, the client has modified the project goal to target MyTab.com to large institutions that utilize multiple login accounts. For the Phase I prototype, the scope has been revamped to target different accounts of Cornell University. Again, the use of frames is unavoidable to have all these accounts open within the same tab. This has been discussed with the client and has been taken forward upon approval from the client on the use of frames.

The current and final scope of the project is to deliver a website which provides users with a master account and the provision to access their Piazza, Blackboard, CMS, Cornell student center, CCNet and MyGannett accounts from within the master account. Users need to add each of these accounts to their MyTab master account by providing their credentials the first time. Once added, these accounts can directly be accessed with just a single click and this frees the users from the hassle of having to type in their different login credentials for multiple accounts every single time.

The website will be implemented in a manner that the design architecture of the system will remain the same for any other institution that the client wishes to extend to in the future. The future developers of the website just need to implement different HTTP communication wrappers for each of them, as each has a different interaction mechanism.

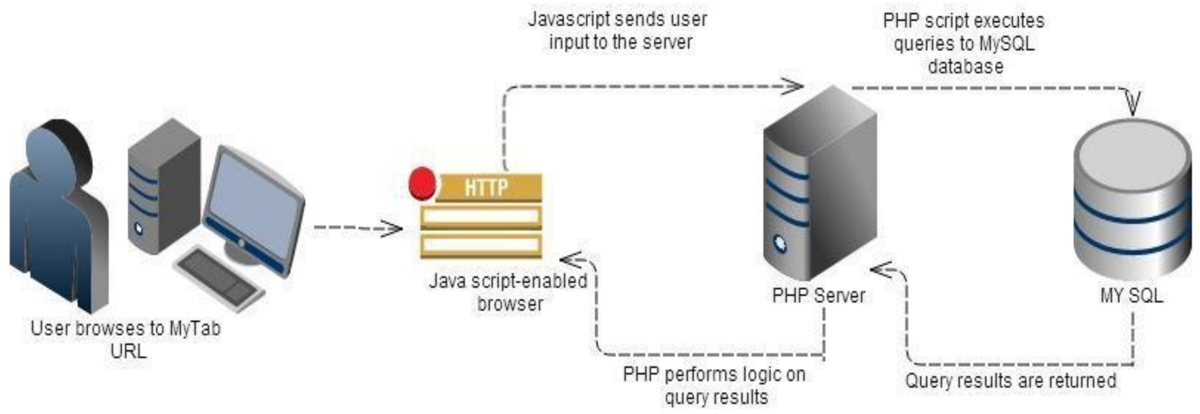
Security is one of the crucial aspects for a website like this. However, the project being only a phase I prototype and not the final production system, tight security will not be implemented within this project. A security framework which needs to be implemented as part of the final production system will be provided.

2. System Design

2.1 System Architecture

Model View Controller

The system will utilize a three-tier architecture, as illustrated in the diagram.

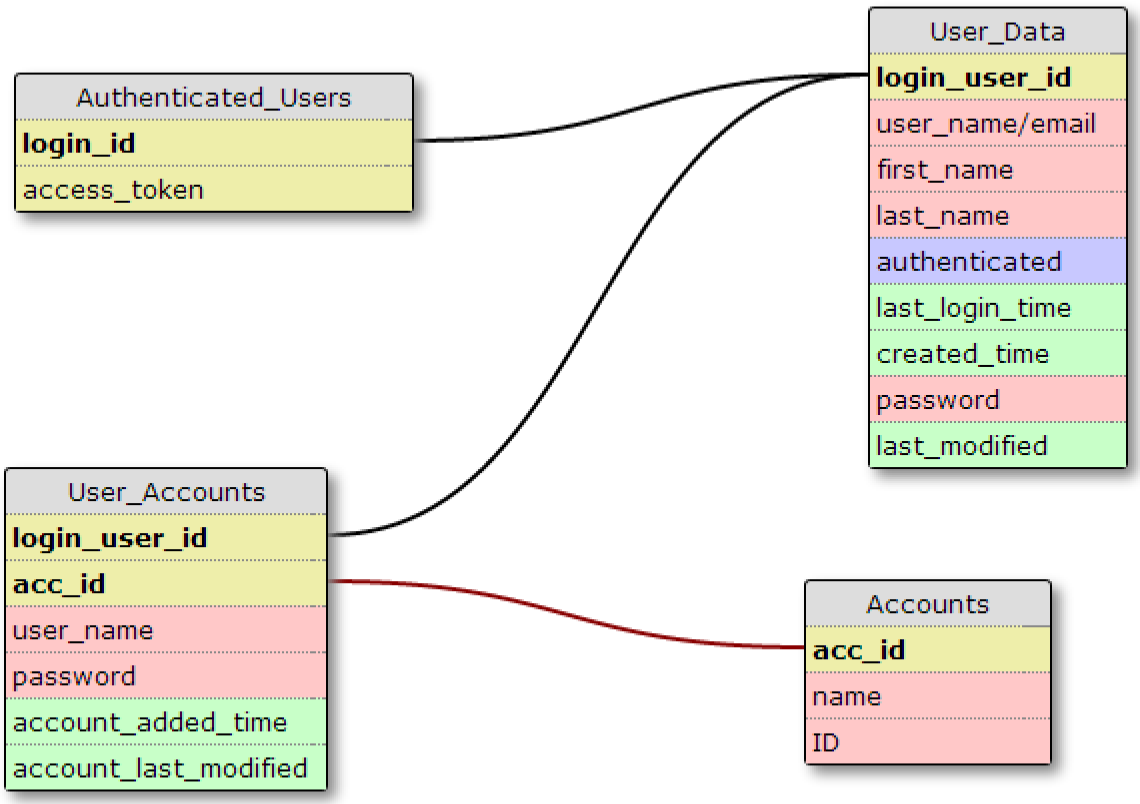

Javascript sends user input to the server
PHP script executes queries to MySQL database
HTTP
User browses to MyTab URL
Java script-enabled browser
PHP Server
MY SQL
PHP performs logic on query results
Query results are returned

The presentation tier will consist of HTML, CSS and Javascript. These technologies are executed by the user’s browser. JavaScript is used on the client side to handle the interactive features. The browser at the client side should be javascript enabled.

PHP 5, a server-side scripting language will be used in the application tier. Querying the database for user and account details and inserting data into the database is handled in this. Dynamic pages are generated based on the user input.

MySQL 5 relational database is used in database tier. The user and all his account details are stored inthis database. The accounts data (image, url, name) is also stored in this database.

2.2DatabaseSystem



These tables represent entities from the previous ER diagrams. This schema is designed in such a way that each table is in 3rd normalized form (3NF). Every user and account is identified by unique id.

3. Program Design

3.1 Overview

Since our application is now contract based, it is pivotal that the architecture has weak coupling among its sub-systems. Broadly there will be a customizable user-interface sub-system, a HTTP communication sub-system and a database tier sub-system.

3.2 User Interface Subsystem

The user-interface component realizes the presentation tier of our application. As per the institution that we contract out our application, the component will be customized according to their needs. Major technologies used here comprise of PHP, HTML, JavaScript and CSS. AJAX has been extensively used to enhance the user experience. This comes under the Model part of our overall MVC architecture. We have a MyTabUIInterface, which lists a basic set of methods that any user-interface created should implement.

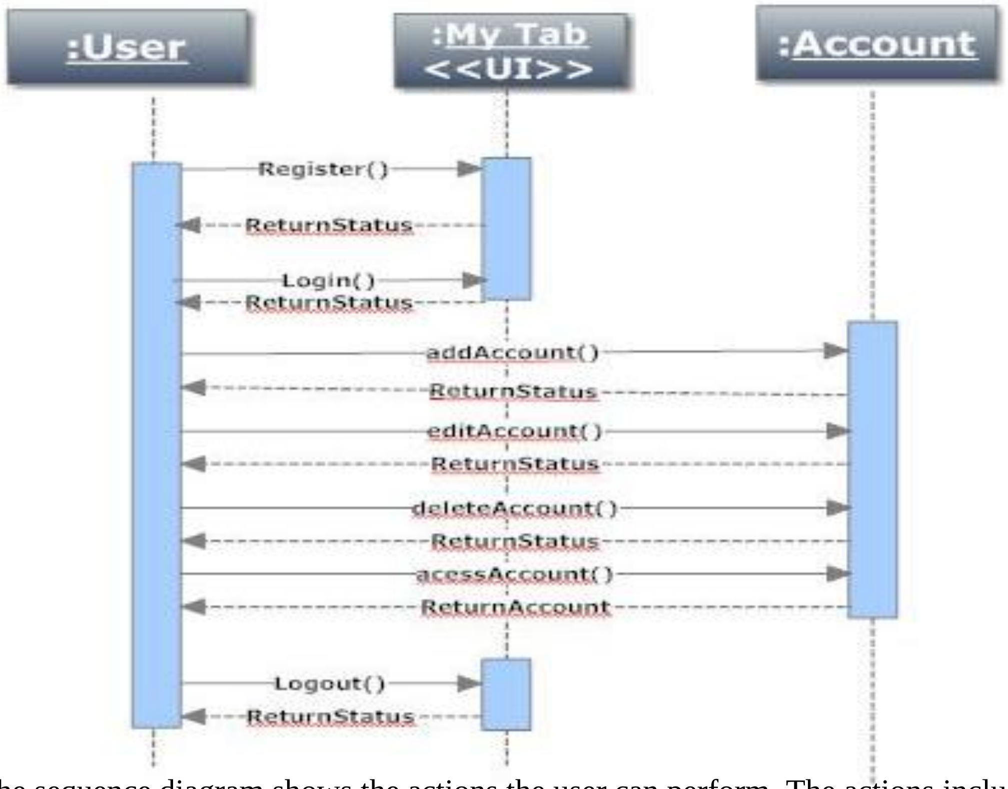
3.3 HTTP Communication and Rendering Subsystem

This component will contain the abstractions required for communicating the web-servers responsible rendering the websites that will be integrated into the MyTab application. This layer will be extensible in-terms of plugging the websites the institution may choose to be a part of this application. For this reason, there will be a Connection interface at the high-level, and all the classes responsible for communicating with the webservers will implement this interface. A basic set of methods, including getting the user credentials from the database, posting the request to the appropriate web-server, etc will be a part of this interface specification.

3.4 Database Subsystem

This allows an abstraction for the database layer to be loosely coupled with the rest of the application. Basically this abstraction will allow the client to easy switch between
the databases he chooses when he install MyTab application.

3.5 Sequence Diagram

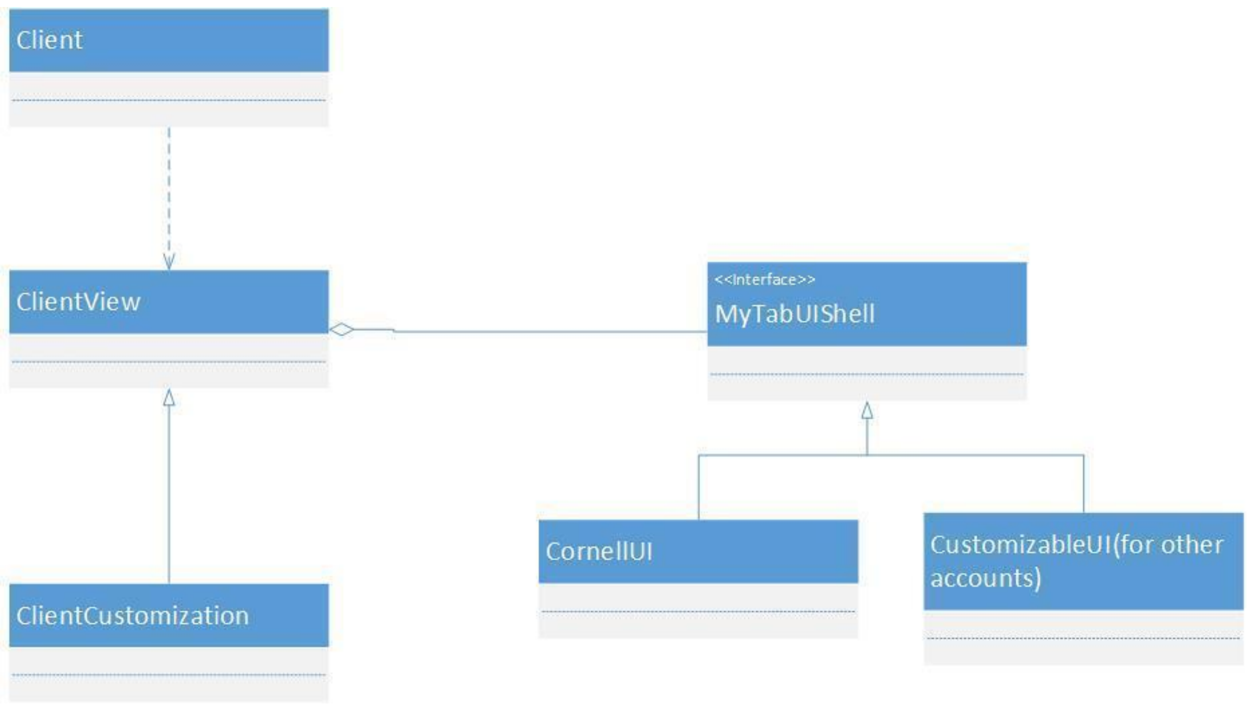


The sequence diagram shows the actions the user can perform. The actions include registering login to the site, add, edit, delete and access the accounts, logout of the site.

3.6 Design Pattern

We will be using the Bridge design pattern to allow the loose coupling among our components and for ability to switch between databases as per the deployment of our system. Bridge design pattern also allows for switching between implementations.We have a generic MyTab UI shell that can be customized according to the needs of the institution through adding new refined abstractions. We have two separate subsystems, one for rendering the appropriate linked account (MyTab would communicate with respective servers to render the accounts), and other for the database tier. Since we are providing user with accounts to navigate through, we will have an Account Rendering Subsystem to allow this. This subsystem allows the client to add/remove accounts according to his needs. A separate database component allows the client to switch between the databases he desires.

High Level Diagram of Bridge Design Pattern Implementation



We are following a Bridge Design pattern for implementing the application. The bridge pattern is applied when there is a need to avoid permanent binding between an abstraction (ClientView) and Implementation (MYTabUIShell) and when the abstraction and implementation need to vary independently. This enables our application to support multiple clients by adding different concrete implementors without disturbing existing concrete implementors. Example currently CornellUI is implemented and in future other clients and institutions can use MyTab without disturbing CornellUI.

ClientView (Abstraction)

Clientviewdefines the abstract interface. Clientviewmaintains the MyTabUIShell reference.

Client Customization (Refined Abstraction)

ClientCustomizationextends the interface defined by ClassView.

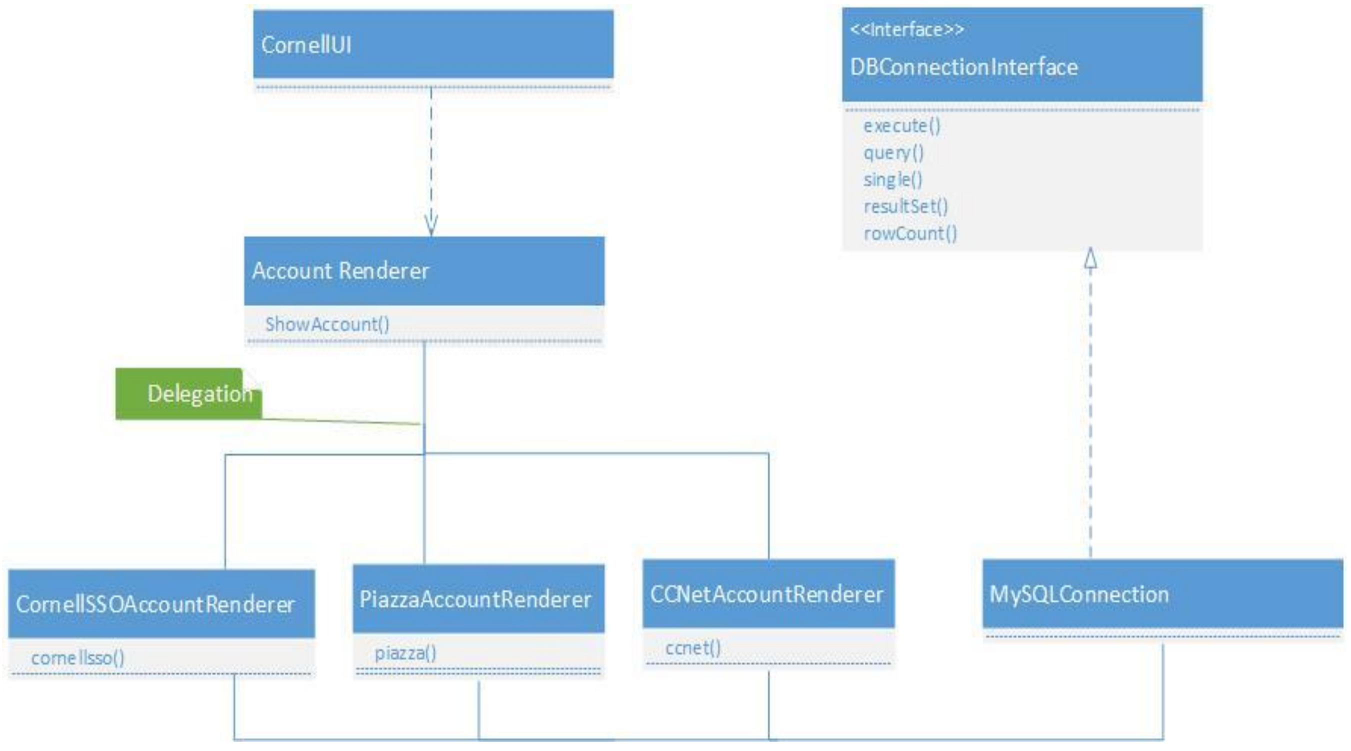
MyTabUIShell (Implementor)

MyTabUIShelldefines the interface for implementation classes. This can be extended by the defined concrete implementation.

CornellUI (ConcreteImplementor)

CornellUIand other concrete implementors implement the MyTabUIShell interface. This concrete implementor depends on the implementation and the client for whom MyTab is being customized.

Design Diagram for Subsystems



For each organization we will have a concrete implementation of the MyTabUIShell interface. Again for each of these concrete implementations, there will be an account renderer which delegates all the accounts that MyTab supports for that organization. This can be seen from the above diagram.

4. Security Architecture

4.1 Analysis

There are two major components of the security design behind MyTab, ensuring secure network transmission and writing secure application code as to prevent client side attacks such as cross site scripting and SQL injections. We discuss both the topics in detail below.

4.2 Secure Network Transmission

For enabling secure data transmission, we have enabled HTTPS on our webserver which will send data securely over SSL. For this purpose we have installed a self-signed SSL certificate in our webserver. By using SSL, we are following industry standard encryption techniques for data transmission. Web browsers give visual cues, such as a lock icon or a green bar, to make sure visitors know when their connection is secured. This means that they will trust the website more when they see these cues and will be more likely to be user of MyTab. SSL providers will also give us a trust seal that instills more trust in the users. SSL layer also protects the user from phishing scams. A phishing email is an email sent by a criminal who tries to impersonate a website. The email usually includes a link to their own website or uses a man-in-the-middle attack to use our own domain name. Because it is very difficult for these criminals to receive a proper SSL certificate, they won’t be able to perfectly impersonate our site. This means that our users will be far less likely to fall for a phishing attack because they will be looking for the trust indicators in their browser, such as a green address bar, and they won’t see it.

We have enabled the SSL layer on our locally installed application using the OpenSSL library. The only problem is that, since the certificate is not actually generated by a certificate authority, the user will get a warning when he visits the application. This is good enough for the prototype, as installing an actual SSL certificate in production server is a fairly trivial process. Also any custom server (Apache) configurations we make should be easily ported to the production server.

A downside is that an SSL certificate is expensive. SSL providers need to set up a trusted infrastructure and validate our identity so there is a cost involved. Because some providers are so well known, their prices can be overwhelmingly high. Performance is another disadvantage to

SSL. Because the information that we send has to be encrypted by the server, it takes more server resources than if the information weren’t encrypted. The performance difference is only noticeable for web sites with very large numbers of visitors and can be minimized with special hardware.

Overall, the disadvantages of using SSL are few and the advantages far outweigh them. It is critical that we properly use SSL on all websites that require sending sensitive information. Proper use of SSL certificates will help protect our customers, help protect us, and help us to gain our customers trust and sell more

4.3 Secure Application Code

Any and all application code written on the server will be tested to vulnerabilities. We use JavaScript for the majority of our client side code. JavaScript enables rich user interactions which would otherwise not be possible. One of the most common JavaScript security vulnerabilities is Cross-site Scripting (XSS), a violation of the same-origin policy. Cross-site scripting vulnerabilities enable attackers to manipulate websites to return malicious scripts to visitors. We have adopted the following this cheatsheet as our guide, drafted by the OWASP project. Some of the points in the website may not be within the scope of this project, but we intend to accommodate the guidelines in this website as much as possible.

We have implemented the “Same origin policy” on our webserver, which will block any malicious scripts originating from a different domain. We also use HTML escaping whenever displaying untrusted data. We also check for an “authentication token” in all of the HTTP request that reach the webserver asking for private information such as user profile details etc. This allows us to avert the "JavaScript hijacking", a type of attack in which a <script> tag on an attacker's site exploits a page on the victim's site that returns private information such as JSON or JavaScript.

Code obfuscation has been implemented to prevent some of the attacks. We are aware of the risk that code can reverse-engineered to understand the logic. We are looking for better ways to perform the obfuscation. Currently we are using YUI Compressor, for our code obfuscation needs.

We implement a strong cookie based authentication and session security mechanism in the website. Since we use cookie based authentication mechanism, there is an added threat of session/cookie hijacking. For this purpose we tie session cookies to the IP address of the user who originally logged in, and only permit that IP to use that cookie. We assign random, nonsequential session IDs and re-authenticate when performing any high-security-critical workflow, such as credential change..etc. This form of re-authentication behavior, when involved changing sensitive information of the user website can be seen in websites such as Facebook or LinkedIn. The re-authentication is not fully in place, and we intend to incorporate it in the next milestone. We also prevent directory listing.

Another important security concern is SQL injections. We perform a strict input validation on the data received from the client and strip it out any unnecessary/malicious input characters before using them in forming SQL queries in the application code. We also implement logical security at the database level; specify users, roles and permissions at the database layer, so that incase if a malicious query is still executed, much of the damage can be averted in the database engine.

4.4 Testing web application security

We are using a combination of open source tools to test the security vulnerabilities of our website. A list of such tools can be found here. As a when we find a vulnerability exposed by one of the tools, we are fixing the code of curb that vulnerability.

Appendix

UI Components

```
Interface MyTabUIInterface{

public function setShellName($name);
   public function getHtml($template);//Outputs the HTML to this UI Shell.

}
```

HTTP and rendering Components

This interface specifies the object creation which contains the details required to making a HTTP request

```
AccountAuthInterface(){
```

```
public function setConnectionURL($url);
    public function setAuthentcationParameters($username,$password);

}
```

This interface specifies the object creation which actually makes the HTTP request and gets back the response required for rendering a linked account.

```
AccountRendererInterface(){

         //$connObj is an object that implements PartnerAuthInterface

public function doGet($connObj);//returns page to be

rendered public function doPost($connObj); //returns page to

be rendered

}
```

The goal behind separating these two is to accommodate the fact that within university, there could be multiple web services which have the same url, username and password. For example, such is the case in cornell single sign on systems.

Database component

```
DBConnectionInterface(){

public function setDBServerURL($url);

public function setDBName($name);

public function setUserName($name);

public function setPassword($name);

}
```

The purpose of this interface is loose coupling with the rest of the application. The client can switch between databases as he chooses.

```
DBSQLInterface(){

public function executeSQLSelect($con,$sql);

public function executeSQLUpdate($con,$sql);

}
```