\*\*APTECH NOTES VIGNESH VK\*\*

$\_SESSION [ ] is a global variable in php . During login. From the users end to keep the user “logged in “ on every page he traverses , a cookie is set. And from the servers end, this global variable is used. Basically can be used as a flag that the user has logged in. once logged out this variable is made empty again

$row is also another global ARRAY variable. Its used to store sql query results in the form of an array

And some imp point I feel like he has told but I cant rmmbr right now. Anyway it stores the query

$row = mysql\_fetch\_array($query) using that. To authenticate user login further u check if $row is empty and then the login session happens.

Also u always need to session\_start(); for login and session end or destroy for lgout . this sets and unsets global variables.

These two are called as session variables..

//SIGN IN WITH GOOGLE SHIZ

All applications follow a basic pattern when accessing a Google API using OAuth 2.0. At a high level, you follow four steps:

#### **1. Obtain OAuth 2.0 credentials from the Google Developers Console.**

Visit the [Google Developers Console](https://console.developers.google.com/) to obtain OAuth 2.0 credentials such as a client ID and client secret that are known to both Google and your application. The set of values varies based on what type of application you are building. For example, a JavaScript application does not require a secret, but a web server application does.

#### **2. Obtain an access token from the Google Authorization Server.**

Before your application can access private data using a Google API, it must obtain an access token that grants access to that API. A single access token can grant varying degrees of access to multiple APIs. A variable parameter called scope controls the set of resources and operations that an access token permits. During the access-token request, your application sends one or more values in the scope parameter.

There are several ways to make this request, and they vary based on the type of application you are building. For example, a JavaScript application might request an access token using a browser redirect to Google, while an application installed on a device that has no browser uses web service requests.

Some requests require an authentication step where the user logs in with their Google account. After logging in, the user is asked whether they are willing to grant the permissions that your application is requesting. This process is called user consent.

If the user grants the permission, the Google Authorization Server sends your application an access token (or an authorization code that your application can use to obtain an access token). If the user does not grant the permission, the server returns an error.

It is generally a best practice to request scopes incrementally, at the time access is required, rather than up front. For example, an app that wants to support purchases should not request Google Wallet access until the user presses the “buy” button; see [Incremental authorization](https://developers.google.com/accounts/docs/OAuth2WebServer#incrementalAuth).

#### **3. Send the access token to an API.**

After an application obtains an access token, it sends the token to a Google API in an HTTP authorization header. It is possible to send tokens as URI query-string parameters, but we don't recommend it, because URI parameters can end up in log files that are not completely secure. Also, it is good REST practice to avoid creating unnecessary URI parameter names.

Access tokens are valid only for the set of operations and resources described in the scope of the token request. For example, if an access token is issued for the Google+ API, it does not grant access to the Google Contacts API. You can, however, send that access token to the Google+ API multiple times for similar operations.

#### **4. Refresh the access token, if necessary.**

Access tokens have limited lifetimes. If your application needs access to a Google API beyond the lifetime of a single access token, it can obtain a refresh token. A refresh token allows your application to obtain new access tokens.

**Note:**Save refresh tokens in secure long-term storage and continue to use them as long as they remain valid. Limits apply to the number of refresh tokens that are issued per client-user combination, and per user across all clients, and these limits are different. If your application requests enough refresh tokens to go over one of the limits, older refresh tokens stop working.

## Scenarios

### **Web server applications**

The Google OAuth 2.0 endpoint supports web server applications that use languages and frameworks such as PHP, Java, Python, Ruby, and ASP.NET.

The authorization sequence begins when your application redirects a browser to a Google URL; the URL includes query parameters that indicate the type of access being requested. Google handles the user authentication, session selection, and user consent. The result is an authorization code, which the application can exchange for an access token and a refresh token.

The application should store the refresh token for future use and use the access token to access a Google API. Once the access token expires, the application uses the refresh token to obtain a new one.

Your application sends a token request to the Google Authorization Server, receives an authorization code,
exchanges the code for a token, and uses the token to call a Google API endpoint.

.

## Token expiration

You should write your code to anticipate the possibility that a granted token might no longer work. A token might stop working for one of these reasons:

* The user has revoked access.
* The token has not been used for six months.
* The user account has exceeded a certain number of token requests.

There is currently a 25-token limit per Google user account. If a user account has 25 valid tokens, the next authentication request succeeds, but quietly invalidates the oldest outstanding token without any user-visible warning.

If you need to authorize multiple programs, machines, or devices, one workaround is to limit the number of clients that you authorize per user account to 15 or 20. If you are a [Google Apps admin](https://support.google.com/a/), you can create additional admin users and use them to authorize some of the clients.

Personal notes: //without backend server authentication wala code.

* In google developers page make sure u use localhost:80 (mention also the port in both javascript origins and redirect page)
* <https://developers.google.com/identity/sign-in/g-normal.png> use this to get the “g” of the google. Site users use from /identity as google is mainsite in the google webpage.

Or make ur own shitty button k? now imma sleep.

Friday, September 04, 2015

New table created

Timestamp section added

Especially for token purposes

CREATE TABLE `users` (

`id` *int*(**11**) NOT NULL AUTO\_INCREMENT,

`oauth\_provider` *varchar*(**255**) COLLATE utf8\_unicode\_ci NOT NULL,

`oauth\_uid` *varchar*(**255**) COLLATE utf8\_unicode\_ci NOT NULL,

`fname` *varchar*(**255**) COLLATE utf8\_unicode\_ci NOT NULL,

`lname` *varchar*(**255**) COLLATE utf8\_unicode\_ci NOT NULL,

userName VARCHAR(40) NOT NULL,

pass VARCHAR(40) NOT NULL

`email` *varchar*(**255**) COLLATE utf8\_unicode\_ci NOT NULL,

`gender` *varchar*(**10**) COLLATE utf8\_unicode\_ci NOT NULL,

`locale` *varchar*(**10**) COLLATE utf8\_unicode\_ci NOT NULL,

`gpluslink` *varchar*(**255**) COLLATE utf8\_unicode\_ci NOT NULL,

`picture` *varchar*(**255**) COLLATE utf8\_unicode\_ci NOT NULL,

`created` datetime NOT NULL,

`modified` datetime NOT NULL,

*PRIMARY KEY* (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8\_unicode\_ci;

Observations:

Backend servers require API’s from github. Each website google, twitter,linkedin,facebook have the repository in the website.

They need to be downloaded and used for the php files.

The database connection and query sent should contain values that are a possibility of being received from the server like first name last name email auth data etc. example

: $insert = mysqli\_query($this->connect,"INSERT INTO $this->tableName SET oauth\_provider = '".$oauth\_provider."', oauth\_uid = '".$oauth\_uid."', fname = '".$fname."', lname = '".$lname."', email = '".$email."', gender = '".$gender."', locale = '".$locale."', picture = '".$profile\_image\_url."', gpluslink = '".$link."', created = '".date("Y-m-d H:i:s")."', modified = '".date("Y-m-d H:i:s")."'") or die(mysqli\_error($this->connect));

}

Make sure u have your redirect url correctly mentioned. Redirect url at the developers console. So when the user is authenticated the php script in ur send will have a redirect url which should be mentioned in the developers console else it does not work properly.

Keep track with the table.

Make sure u revoke token access from logout.

For some reason the port of the xampp localhost also plays a role in mentioning in the developers console. (google)

Imma sleep now gnite.