# HTTP/1.1 Authentication June 2014

# Header Field Definitions

* **WWW-Authenticate (response header)**
* **Authorization**

The "Authorization" header field allows a user agent to authenticate

itself with an origin server -- **usually, but not necessarily, after**

**receiving a 401 (Unauthorized) response**. Its value consists of

credentials containing the authentication information of the user

agent for the realm of the resource being requested.

Authorization = credentials

If a request is authenticated and a realm specified, the same

credentials are presumed to be valid for all other requests within

this realm (assuming that the authentication scheme itself does not

require otherwise, such as credentials that vary according to a

challenge value or using synchronized clocks).

A proxy forwarding a request MUST NOT modify any Authorization fields

in that request.

**Two different ways of authentication: stateful and stateless authentication.**

* **Stateful authentication**

Stateful authentication is commonly used in many applications, especially for applications that do not require scalability too much.

How it works

**One way: Cookies**

Stateful session is created on the backend side, and the corresponding **session reference Id** is sent to the client. Each time the client makes a request to the server, **the server locates the session memory using the reference Id from the client and finds the authentication information**.

In this model, you can easily imagine that if the session memory is deleted on the backend side, then the session reference Id, which the client is holding, is completely meaningless.

## Advantages

* **Revoke the session anytime**
* **Easy to implement and manage for one-session-sever scenario**
* **Session data can be changed later** (assume that for a one-session-sever, no inconsistent problem)

## Disadvantages

* **Increasing server overhead**: As the number of logged-in users increases, the more server resources are occupied.
* **Fail to scale**: If the sessions are distributed in different servers, we need to implement a tracking algorithm to link a specific user session and the specific session sever. That means, once Bob’s session is handled by X server, then all Bob’s following requests must be handled by X server. This can be done by adding a tag to the client request in the proxy layer (e.g. HAProxy) before routing to the backend. The proxy layer using this tag to determine which backend server to route.  
  Furthermore, if the session servers are deployed with duplication to have fail-over, the problem becomes more complicated since the 2+ peers duplicating each other must implement an algorithm to make sure the consistency of their sessions.
* **Difficult for 3rd party applications to use your credentials**: In fact, this is also one of the “fail to scale” example, but I note it down separately since it is important. When a 3rd party application enables your users to login their website, the 3rd party application is not able to directly verify your users’ session (they are stored on your backend). The verification must be redirected to the credential servers. Therefore, there is more work between 3rd party application and the backend.

# Stateless authentication

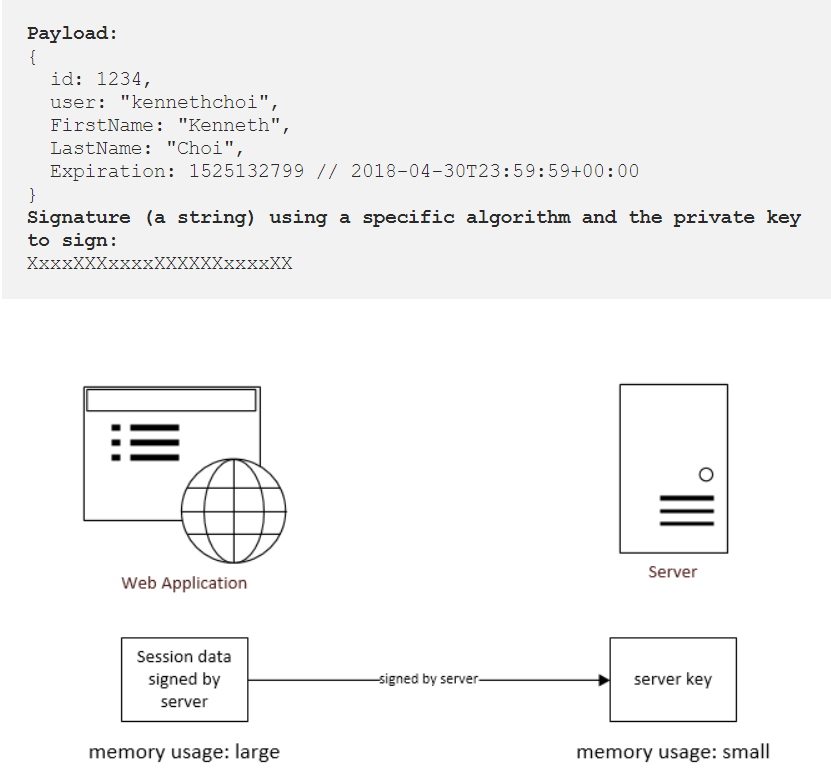
Stateless authentication is used to solve the disadvantages of stateful authentication. They are quite different and are used in different scenarios.

## How it works

If you are looking for concrete industrial standards, please refer to **OpenID Connect** and **JSON Web Token (JWT)**.

Stateless authentication stores the **user session data** **on the client side** (browser). The data is signed by the key of IdP to ensure the integrity and authority of the session data.

Since the user session is stored on the client side, the server only have the capability to verify its validity by checking whether **the payload and the signature match**.



## Advantages

* **Lower server overhead**: The great number of session data does not store on the server side. We can store more user properties on the client-side session data to reduce the number of database access without worrying the memory overhead on the server.
* **Easy to scale**: Since the session data is stored on the client side, it does not matter which backend server the request is routed to, **as long as all backend servers share the same private key**, then all servers have the same capability to verify the validity of the session.
* **Good to integrate with 3rd party application**: In the single sign-on protocols, the 3rd party applications and the IdP must be able to communicate with each other via user agents (browser). During the account linking process between 3rd party applications and IdP, IdP sends a signed message to the browser, and the browser redirects this message to the 3rd party applications. Using a pre-configured shared secrete, the 3rd party applications can determine whether the account linking (single sign-on) is valid by itself.

## Disadvantages

* **Cannot revoke the session anytime**: Since the user session is stored at client side, the server does not have any rights to delete the session.
* **Relatively complex to implement for one-session-server scenario**: The advantages of stateless authentication is scalability. However, it increases the technical complexity and it is not extremely useful when we only have one-session-server.
* **Session data cannot be changed until its expiration time**: Suppose we want to add “Age” property to the session data above, probably we can ask the client to update it, but we cannot make sure the client does update it, since its previously session data is not expired yet, then the client still has the chance to make requests with old session data.

# Remember-Me authentication

* Remember-me or persistence-login authentication allows websites to remember the identity of a logged-in user between sessions.  **Spring Security sends a cookie to browser, when user login to the application with remember-me option. This cookie is stored at browser side for a specific time period.** Next time, when user logins to application, Spring security will check and validate the stored cookie and cause the automatic login if cookie is valid.
* Spring Security provides two approaches for implementing remember-me -
* **Hash-Based Token Approach** - In this approach, username, expiration time, password and a private key are hashed and sent to browser as a token.
* **Persistent Token Approach** - In this approach, a database or other persistent storage mechanism is used to store the generated tokens.

Hash-Based token approach has a potential security issue because it contains the password in hash string.

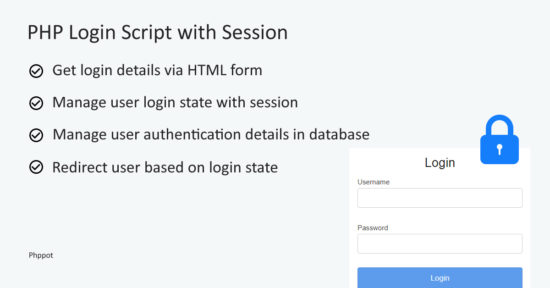
* This example will help you [to build a persistent](https://phppot.com/php/how-to-build-a-persistent-shopping-cart-in-php/) authentication system for your PHP web application. When the user attempts to log in with the application, the entered login credentials are verified with the database. If a match is found, the PHP session and **the cookies are used to preserve user logged-in state** before **redirecting the user to the dashboard.** On successful login, the unique member id from the member database is stored in a session. Then, **the cookies are set to keep the login name and the password for a specified expiration period**.  Instead of storing the users’ plain password, random password and token are generated and stored in the cookie to avoid hacking.
* When the user accessing the application pages, the [existing logged in session](https://phppot.com/php/php-login-script-with-session/) is checked to redirect the user to access the requested page. If the session is empty, then the code will check the logged-in with the cookies. If both the session and the cookies are not having any data about the remembered login, **then the user will be redirected back to the login page**. The authentication cookies are set with the expiration time of 1 month. The random password and tokens will be stored in the database with the expiration date and time. The cookie-based logged in state validation is done by testing cookie availability and expiration stored in the database.

1. **PHP Login Script with Session**

Last modified on August 12th, 2019.

In this tutorial, let us create a login script with a session in PHP. It has a simple example of implementing user authentication. This example uses a standard login form to get the user login details. And it preserves the login state with PHP sessions.

Login would be the first step of many application. Sometimes, part of privileged functionalities of application will ask users to login.



**Ways to create an authentication system**

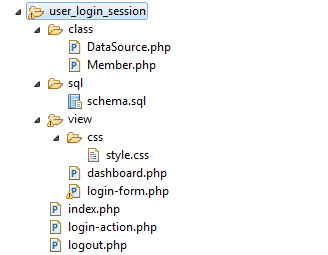
There are different ways of implementing an authentication system. The most popular way is to get the username and password via a login form and authenticate based on them.

1. **Basic Authentication with username, password Login**

* In this example, it has **users database** with name, email, password and more details. It has a HTML form with inputs to get the user login credentials.
* When the user submits their login details, then the PHP code will receive the posted data. It compares the entered data against the user database.
* If match found, then it sets the user login session. In this authentication code, it preserves the user id in a PHP session. The existence of this session will state user authentication status.
* After authentication, the PHP $\_SESSION [super global variable](https://phppot.com/php/php-globals-with-egpcs-information) will contain the user id. That is, the $\_SESSION[“member\_id”] is set to manage the logged-in session. It will remain until log out or quit from the browser.
* While logout, we unset all the session variables using [PHP unset() function](https://phppot.com/php/php-unlink-vs-unset/).

Below screenshot shows the organized file structure of this user login example. The Member.php is the model class with authentication functionalities.

* The DataSource.php file contains functions to get a connection and access database.
* In a view directory, I have created all the UI related files for the login and the dashboard interface. It also contains a stylesheet used for this UI.
* The index.php is the landing page that checks the user logged-in session. Then it redirects users either to log in or to the dashboard.
* The login-action.php and logout.php files are the PHP endpoints. They handle actions as requested by the users via the interactive authentication Interface.

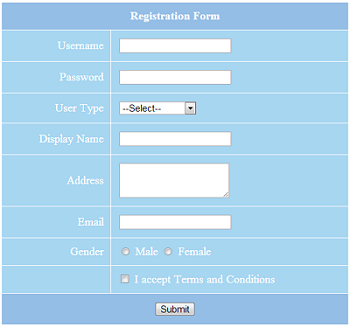


### **Login form validation from client side**:

This script is for validating the login data at the client-side. If the users submit the login with empty fields then this script will return boolean false.

When it returns false, it displays a validation error message to the users. By returning boolean 0, the [form validation script](https://phppot.com/php/php-form-validation/) prevents login to proceed further.

* While Validating an HTML form the client side validation is not only enough. Since our web application allows outsiders to enter data, server-side form validation is also required for additional security.
* Here, we are going to see how to validate a form using PHP. We have a registration form containing name, password, email and gender fields. All the fields are mandatory and should not empty. The email should be in correct format. We are going to check these validations in server side.



## Prepopulate Form with Validation Error Message.

Once the form contains invalid data then this code pre-populates the form with invalid data and display validation errors.