# WebFrontAuth configuration

The WebFrontAuthMiddleware is configured by a classical AuthenticationOption class:

public class WebFrontAuthMiddlewareOptions : AuthenticationOptions

{

/// <summary>

/// The <see cref="WebFrontAuthMiddleware"/> is not designed to be added multiple

/// times to an application, hence its name is unique.

/// </summary>

public const string OnlyAuthenticationScheme = "WebFrontAuth";

/// <summary>

/// Initializes a new instance of <see cref="WebFrontAuthMiddlewareOptions"/>.

/// </summary>

public WebFrontAuthMiddlewareOptions()

{

AuthenticationScheme = "WebFrontAuth";

AutomaticAuthenticate = false;

AutomaticChallenge = false;

}

/// <summary>

/// Gets the entry point: "/.webfront".

/// </summary>

public PathString EntryPath => \_entryPath;

/// <summary>

/// Controls how much time the authentication will remain valid

/// from the point it is created.

/// Defaults to 20 minutes.

/// This time is extended if <see cref="SlidingExpirationTime"/> is set and

/// when "<see cref="EntryPath"/>/c/refresh" is called.

/// </summary>

public TimeSpan ExpireTimeSpan { get; set; } = TimeSpan.FromMinutes( 20 );

/// <summary>

/// Controls how much time the long term, unsafe, authentication information

/// will remain valid from the point it is created.

/// Defaults to one year.

/// </summary>

public TimeSpan? UnsafeExpireTimeSpan { get; set; } = TimeSpan.FromDays( 366 );

/// <summary>

/// Gets whether <see cref="UnsafeExpireTimeSpan"/> is not null, greater than <see cref="ExpireTimeSpan"/>,

/// and <see cref="CookieMode"/> is not <see cref="AuthenticationCookieMode.None"/>.

/// When true a long-lived cookie is used to store the unsafe, but long term, authentication information.

/// Its <see cref="CookieOptions.Path"/> depends on <see cref="CookieMode"/>.

/// </summary>

public bool UseLongTermCookie => UnsafeExpireTimeSpan.HasValue

&& UnsafeExpireTimeSpan > ExpireTimeSpan

&& CookieMode != AuthenticationCookieMode.None;

/// <summary>

/// Gets whether the authentication cookie (see <see cref="CookieMode"/>) requires or not https.

/// Note that the long term cookie uses <see cref="CookieOptions.Secure"/> sets to false since it

/// does not require any protection.

/// Defaults to <see cref="CookieSecurePolicy.SameAsRequest"/>.

/// </summary>

public CookieSecurePolicy CookieSecurePolicy { get; set; }

/// <summary>

/// Gets or sets if and how cookies are managed to store the authentication information.

/// <para>

/// Defaults to <see cref="AuthenticationCookieMode.WebFrontPath"/>.

/// </para>

/// <para>

/// Setting it to <see cref="AuthenticationCookieMode.RootPath"/> should NOT BE used for

/// professional development: this mode, that is the same as the standard Cookie ASP.Net authentication,

/// works only for standard and classical Web application.

/// </para>

/// <para>

/// Setting it to <see cref="AuthenticationCookieMode.None"/> disables all cookies: client apps

/// are no more "F5 resilient", this can be used for pure API implementations.

/// </para>

/// </summary>

public AuthenticationCookieMode CookieMode { get; set; }

/// <summary>

/// Gets or sets a list of available schemes returned for information from '/c/refresh' endpoint

/// when 'schemes' appears in the query string.

/// <para>

/// Defaults to null: schemes are the same as <see cref="IWebFrontAuthLoginService.Providers"/>

/// when this is null or empty.

/// </para>

/// <para>

/// When not null (or empty), this list takes precedence over the login service's providers: all supported

/// schemes must be declared here (and unwanted ones must not appear).

/// </para>

/// <para>

/// This list does not forbid user login to non listed schemes, this is intended only for applications

/// to communicate with the user..

/// </para>

/// </summary>

public List<string> AvailableSchemes { get; set; }

/// <summary>

/// Gets or sets a function that may allow calls to '/c/unsafeDirectLogin' for schemes.

/// Enabling calls to to this endpoint must be explicit: no configuration means "403 - Forbidden".

/// </summary>

public Func<HttpContext, string, bool> UnsafeDirectLoginAllower { get; set; }

/// <summary>

/// Gets or sets the refresh validation time.

/// When set to other than <see cref="TimeSpan.Zero"/> the middleware will re-issue a new token

/// (and new authentication cookie if <see cref="CookieMode"/> allows it) with a new expiration time any time it

/// processes a "<see cref="EntryPath"/>/c/refresh" request.

/// This applies to <see cref="IAuthenticationInfo.Expires"/> but not

/// to <see cref="IAuthenticationInfo.CriticalExpires"/>.

/// </summary>

public TimeSpan SlidingExpirationTime { get; set; }

}

# WebFrontAuth protocol

WebFrontAuth middleware handles all requests that start with /.webfront/.

## (POST) .webfront/c/basicLogin

This entry point supports basic authentication via user name and password.

Request body:

{

"userName": "Albert",

"password": "pwd"

}

Response :

* If IWebFrontAuthLoginService.HasBasicLogin is false   
  🡺 404 Not Found.
* If the request body is not valid   
  🡺 400 Bad Request.
* If basic authentication fails   
  🡺 401 Unauthorized
* If basic authentication succeeds   
  🡺 200 OK

{

"info": {

"user": {

"id": 2,

"name": "Albert",

"providers": [

{

"name": "Basic",

"lastUsed": "2017-07-26T14:50:48.5767287Z"

}

]

},

"exp": "2017-07-26T15:10:58.7503983Z"

},

"token": "CfDJ8CS62…pLB10X",

"refreshable": false

}

When refreshable is true, calls to c/refresh should be done before exp to refresh the expiration date.

## (GET, POST) .webfront/c/refresh[?schemes]

This should be the first call from a client that starts without any context: if cookies exist they are used to restore the authentication token. The other use of this entry point is to refresh an existing token before its expiration.

* If a valid authentication is found (non-expired Authorization token or Authorization cookie)   
  🡺 200 OK

Response body is the same as the return of a successful authentication with a possibly updated expiry date.

* If no valid authentication is found but the long-lived cookie exists  
  🡺 200 OK  
  There is no exp field: this is an **unsafe** user information.

{

"info": {

"user": {

"id": 2,

"name": "Albert",

"providers": [

{

"name": "Basic",

"lastUsed": "2017-07-26T15:17:58.9615326Z"

}

]

}

},

"token": "CfDJ8CS6...H9vQeL7NZa1Aywib0NJ69X-",

"refreshable": false

}

* If no authentication at all has been found (or the Authorization token is invalid)  
  🡺 200 OK

{

"info": null,

"token": null,

"refreshable": false

}

The ?schemes query string parameters adds a schemes field to the response body that is an array of the available scheme names.

## (GET, POST) .webfront/c/logout[?full]

Logout the user by removing the authentication cookie and, if ?full query parameter is specified, the long-lived cookie.

## (GET, POST) .webfront/c/startLogin?scheme=SCHEME[&returnUrl=/PAGE]

Starts an external login process. SCHEME identifies the authentication scheme to use. By default, the process is done in a popup window but if returnUrl is specified, the process uses redirections (the web client application context is lost).

Client applications can easily transfer information through the process:

* In inline mode (returnUrl)
  + The parameters of the returnUrl are kept.
* In default mode (popup)
  + When using GET verb, any query string parameters that are not scheme nor returnUrl are considered as userData.
  + When using POST verb, any form data are considered as userData.

Examples:

* Using the inline mode:

GET /.webfront/c/startLogin?scheme=oidc&returnUrl=/auth-done?p=67

The process ends with this last response:

<!DOCTYPE html>

<html><body><script>

(function(){window.url='http://localhost:4324/auth-done?p=67';})();

</script></body></html>

* Using the default mode:

GET /.webfront/c/startLogin?scheme=oidc&A=3&A=p&Other=param&X

The process ends with this last response:

<!DOCTYPE html>

<html>

<body>

<script>

(function(){

window.opener.postMessage( {"info":{"user":{"id":3,"name":"carol","providers":[{"name":"Oidc","lastUsed":"2017-07-26T16:21:12.89Z"},{"name":"Basic","lastUsed":"2017-07-26T16:19:38.51Z"}]},"exp":"2017-07-26T16:41:12.9034089Z"},"token":"CfDJ8CS62t…Aoa93HA","refreshable":false,"initialScheme":"oidc","callingScheme":"oidc","userData":{"A":["3","p"],"Other":"param","X":""}}, 'http://localhost:4324/');

window.close();

})();

</script>

<!--7Mq…KJHyzsdHI/-->

</body>

</html>

## (GET, POST) .webfront/token

This is the only entry point that is not subordinated to the /c/ path: by default cookies cannot be used since they are only available below /.webfront/c/.

This acts as any “normal” API and is an easy way to test the authentication. It always returns the Authentication information in JSON (it will be able to use the authentication cookies only if WebFrontAuthMiddlewareOptions.CookieMode is AuthenticationCookieMode.RootPath just like any other API on the server).