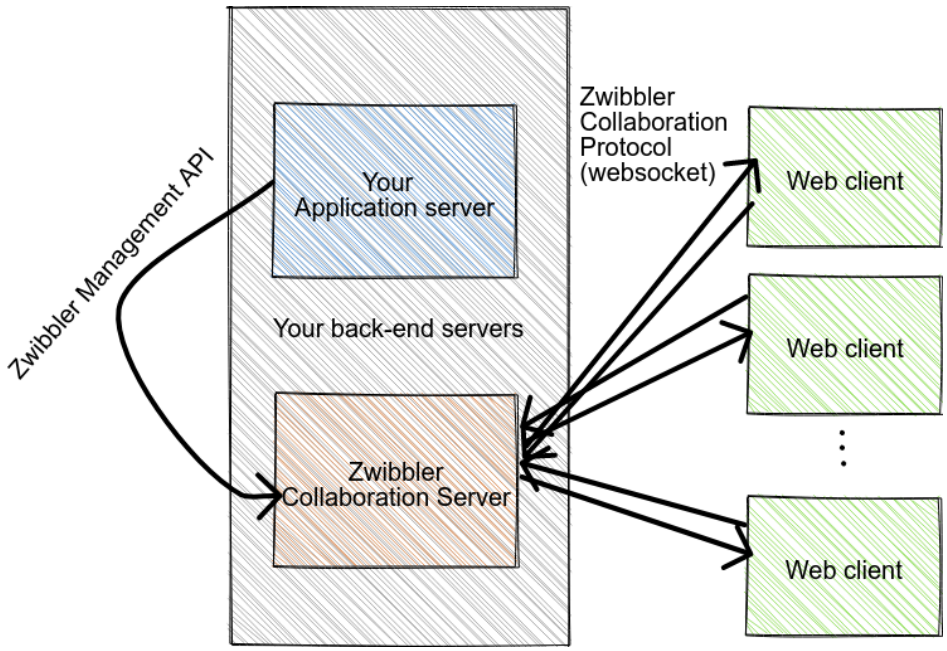
Zwibbler collaboration server Management API v0.1

This document describes the Zwibbler Collaboration Server Management API (MAPI). The functions of the management are:

- Provide secure access to the sessions, so that only authorized users may connect to a session, and they can have separate view and modify, and "admin" permissions enforced at the server level.
- Dump, delete, and create documents[1].
- Provide notification when various events have occurred.

The MAPI refers to communication between your application server and your private instance of the Zwibbler Collaboration Server. Communication between the web clients and the collaboration server is described in Zwibbler Collaboration Server Protocol V2.



[1] A session refers to a document, or room, to which users can connect and modify. These words are used interchangeably in this document.

Permissions

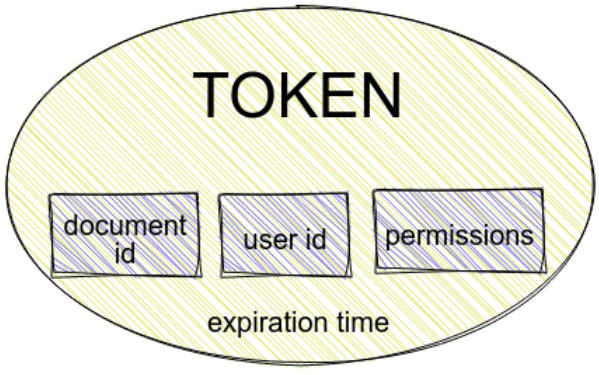
Server level security allows three different classes of users, restricting what they can do with whiteboards.

Since setting key/value pairs is used for presence information, all classes of users may set these values, but certain key names are limited to administrative class users.

Permissions	Examples	Description
"r"	Students in a class	• Can see changes • Can set general keys
"rw"	Students in a class temporarily allowed to write to a board	• Can see changes • Can make changes • Can set general keys
"rwa"	Teachers or moderators	• Can see changes • Can make changes • Can set general keys • Can set keys with names beginning with "admin:"

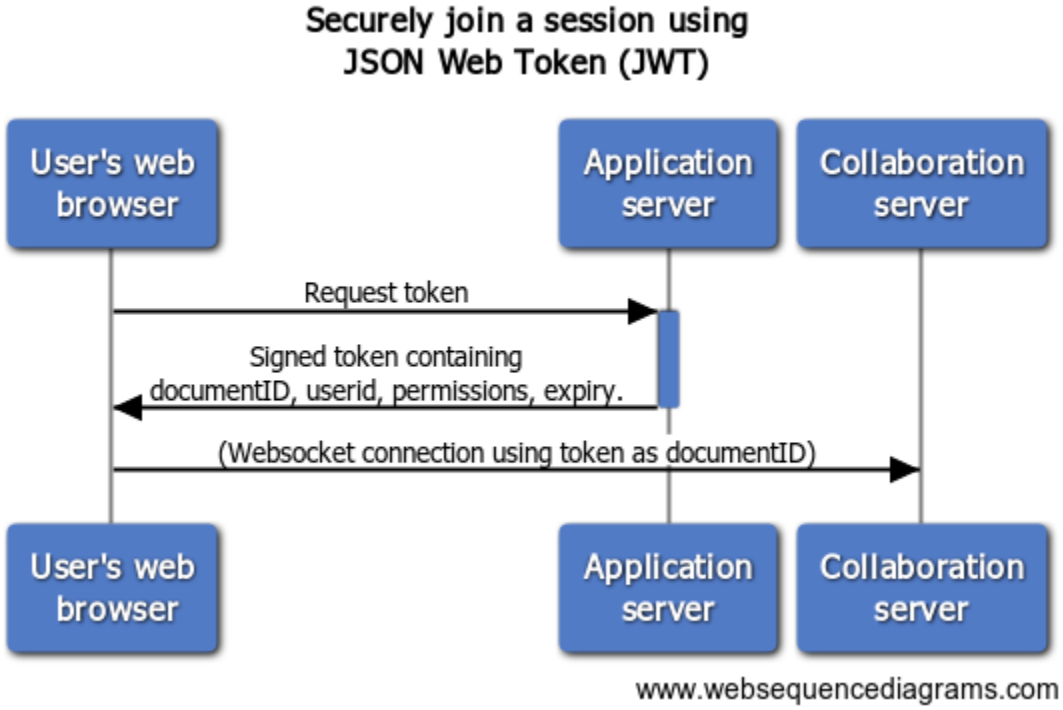
Tokens

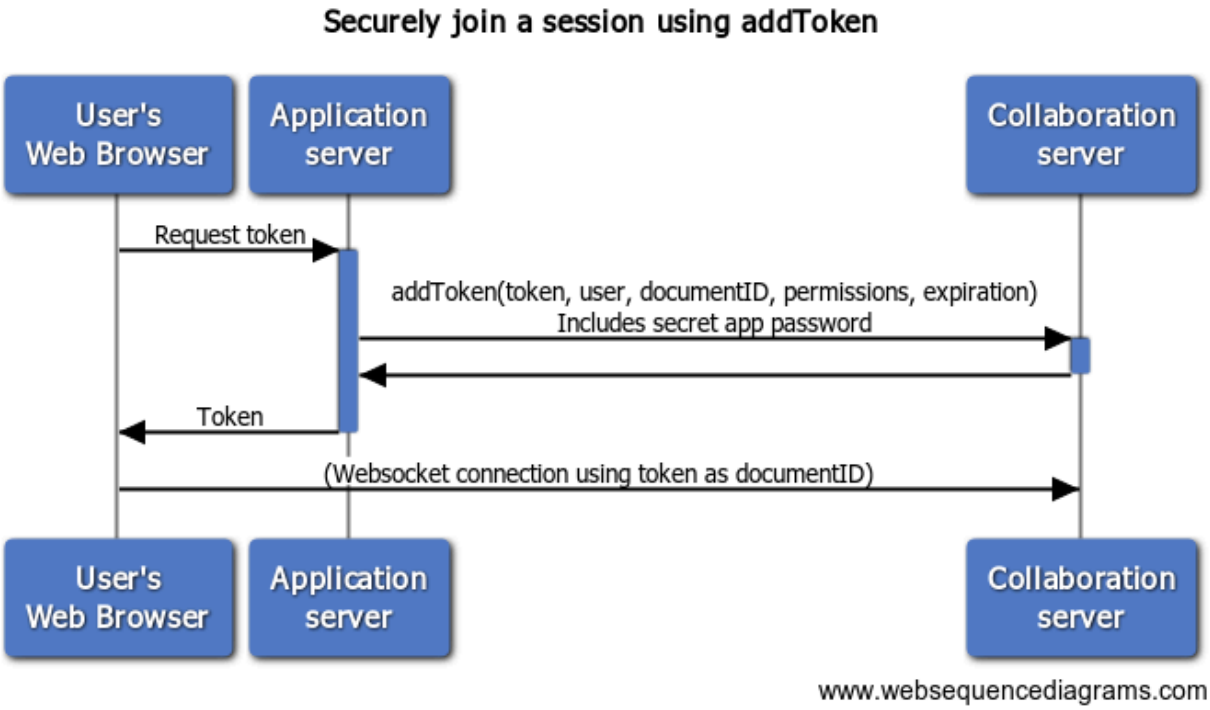
Please refer to these example flows as you read the rest of the document. The central idea is that instead of the clients connecting to a document using its ID, they instead use a different identifier that is unique to each client, called a *token*. The collaboration server will then look up the correct document from this token.

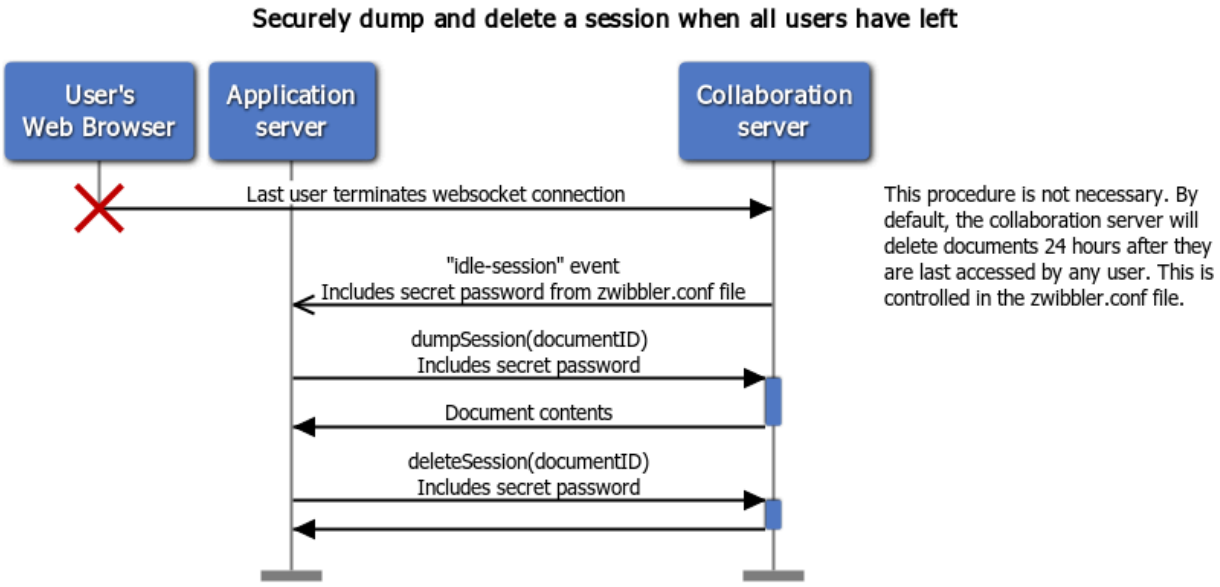


The token associates a particular user with a document ID and the user's read/write permissions. Tokens automatically expire after a period of time.

This token can either be a signed JSON Web Token, or any string you choose. In the second case, it must be pre-configured by your application server before the user tries to connect, using a call to addToken().







Security between application server and collaboration server

The application server refers to the backend that you implement. It is separate from the collaboration server. Your application server may wish to delete sessions or dump their information for long-term storage. This level of access and control requires authentication.

For this purpose, we use HTTP Basic authentication. A username / password is configured in the **Collaboration server configuration file** (usually zwibbler.conf). Requests for server management functions must contain this username/password, otherwise they will fail.

If you use the webhooks feature, you should also authenticate the call from the collaboration server by checking if the HTTP username/password matches the one configured.

Endpoint

For simplicity in configuring web server forwards, the collaboration server uses a single endpoint for both websocket and API calls, which is usually "/socket". This endpoint is used in different ways.

How it is accessed	**Result**
GET request to /socket and no other parameters	An HTTP response with the text "Zwibbler collaboration Server is running." This is important to verify the initial setup of the collaboration server.
GET request to /socket and an Upgrade field in the HTTP	This is how Zwibbler's client library connects to the server over a websocket connection. The server will then wait

headers	for the INIT message for a period of time, as described in the Zwibbler Collaboration Protocol document. If secure access is enabled on the server, the document ID of the INIT message refers to a token, instead of a document id.
POST request with a "method" and other parameters.	The request is treated as a session management request. It must contain a Base64-encoded username/password as part of the HTTP Basic authentication headers.

Session management

With MAPI, each user is forced to use a token instead of an explicit document identifier. The token associates together the document id, user, and permissions. There are two ways of getting these tokens to the server.

Implicit JSON Web Token	**Explicit tokens**
Tokens are created and signed by your application server, and sent to the client. When the client uses them, they are verified using a pre-shared key and implicitly trusted.	Tokens are identifiers created by your application server. Before being used, your application server must first contact the collaboration server using a POST call to addToken, and then give that token to the client for use.

Session management using implicit JWT

To create a JWT token, you first configure the server with a SHA256 key in the configuration file. You will use this key to create the token using a JWT library of your choice.

The token must contain the following claims:

Claim	Description
exp	The expiration time of the token, as a numeric value of the number of seconds since the epoch (unix time).
sub	The document ID as a string. The "sub" stands for subject, and is chosen because it is a standard claim in the JWT specification.

u	The userid as a string.
p	The permissions, as a string. See the documentation for addToken below.

When JSON Web Tokens are used, the application server does not need to call addToken before using them.

Session management using explicit tokens

Session management refers to the creation and deletion of sessions and tokens. All of these REST methods are expected to be between the Application Server and the Collaboration server, never from a Web browser. That way, HTTP Basic authentication can be used.

To call these methods, make a POST request. The URL must be the endpoint used by the server (usually ending in /socket). The username and password must be encoded in the Authorization field of the HTTP headers. The parameters are sent using x-www-form-url-encoding.

addToken

This method is deprecated because it is not compatible with redis-cluster. You should use JWT instead.

POST parameter	**Description**
method	"addToken"
token	New token identifier - a text string.
documentID	The document ID - a text string.
userID	A user id - a text string used to identify users in updateUser
permissions	"r" - read only "rw" - read and write. "rwa" - read, write, and set keys beginning with "admin:" "" - user may not access whiteboard.
expiration	Expiration time, as an RFC date using the same format as an HTTP cookie.
contents	(Optional) contents of the document to create

Adding a token creates an association between a particular user, the document, and permissions. Optionally it can be used to create a new document from previously saved data. If contents are specified and the document already exists, the call will fail with HTTP error code 409 (Conflict).

If the token already exists, HTTP error code 409 (Conflict) will be sent.

If the username/password in the authentication headers do not match the ones from the configuration file, HTTP code 401 will be returned.

updateUser

POST parameter	**Description**
method	"updateUser"
userID	The user id, as specified in a previous call to addToken or inside a JWT token
documentID	The document ID.
permissions	New permissions (See addToken)

This method changes the permissions for a particular user. If that user is not connected to the given document and has no unexpired tokens, it has no effect.

If permissions is "", the user is immediately disconnected from the whiteboard.

deleteDocument

POST parameter	**Description**
method	"deleteDocument"
documentID	The document ID.

When a document is deleted, all users viewing that document are immediately disconnected and the document is removed from the database.

dumpDocument

POST parameter	Description
method	"dumpDocument"
documentID	The document ID.

RESPONSE

The response is the text of the document. This text can be directly opened in Zwibbler using ctx.load() or used in a call to createDocument on the collaboration server. If it does not exist, HTTP 404 is returned.

checkDocument

This was been added in October in 2023

POST parameter	Description
method	"checkDocument"
documentID	The document ID.

RESPONSE

This returns an empty response. The response code is 404 if the document does not exist and 200 if it exists.

createDocument

POST parameter	Description
method	"createDocument"
documentID	The document ID.
contents	The contents of the document. This parameter may be sent in the form data as either a text field or a file. When encoded as a file, the name and content-type are ignored.

If the document already exists, HTTP 409 (Conflict) is returned

Events

The server may be configured to notify an endpoint when all users have left a session. In this case, the server will make an HTTPS request to the endpoint, including a username/password specified in the configuration file, containing data about the event.

The request will be a POST request with the following parameters, in addition to the HTTP Basic authentication in the headers.

POST parameter	**Description**
event	"idle-session"
documentID	The identifier of the document.

Test cases

addToken()

HTTP 401 is sent if username / password do not match
HTTP 400 is sent if missing fields or parameters are incorrect.
HTTP 409 is sent if token already exists
HTTP 409 is sent if contents are specified and document already exists
After calling it, users can connect to the document using the token
After expiry, users can no longer connect to the document using the token.

updateUser()

HTTP 401 is sent if username/password do not match
HTTP 400 is sent if parameters are missing / incorrect
The user receives special NACK code 2 if permissions are changed to readonly and he/she tries to alter the document
The user is immediately disconnected if the permissions are changed to ""

deleteDocument()

HTTP 401 is sent if username/password do not match
HTTP 400 is sent if parameters are missing / incorrect
All users receive error 0x0001 and are immediately disconnected from the document
After deleting a document, it is no longer available using dumpDocument()

createDocument()

HTTP 401 is sent if username/password do not match
HTTP 400 is sent if parameters are missing / incorrect
If the document already exists, HTTP 409 (Conflict) is returned
After creating the document, it is available using dumpDocument

checkDocument()

HTTP 401 is sent if username/password do not match
HTTP 400 is sent if parameters are missing / incorrect
HTTP 404 is sent if the document does not exist
HTTP 200 is sent if the document exists

dumpDocument()

HTTP 401 is sent if username/password do not match
HTTP 400 is sent if parameters are missing / incorrect
HTTP 404 is sent if the document does not exist

* Success case is tested as part of createDocument()

Permissions

After an INIT message, if the user has no read permissions, then the error is "access denied" and disconnection.
If the user lacks write permission, and the creation mode is ALWAYS CREATE, the error is "access denied".

If the user lacks write permission, and the creation mode is POSSIBLY_CREATE and the document does not yet exist, the error is "access denied".
If the user lacks write permission and the creation mode is POSSIBLY_CREATE and the document already exists, they receive the contents of the existing document.
Upon receiving the APPEND message, if the user lacks write permissions, the server shall respond with ACK/NACK code 0x0002
When connecting with administrative permission, user can set keys beginning with admin:
When connecting without administrative permission, when setting a key beginning with "admin:" then user receives SET_KEY_ACK with NACK=1 and other users do not receive keys.

JWT

Expired JWT in INIT message is rejected with "access denied" and disconnection.
JWT with bad signature is rejected with "access denied" and disconnection.
Correctly formatted JWT is accepted.
When JWT is used, clients must use a token and cannot create a document using a random identifier.