twitch.json

misinformation

Page ID: 84

Source: https://dev.twitch.tv/docs/extensions/required-technical-background/

['authentic']

Required Technical Background | Twitch

Developers Asset 20 Products Showcase Support Blog Docs Tutorials Log in with Twitch Contents Overview Twitch API EventSub Chat & Chatbots Authentication Organizations Drops Embedding Twitch Extensions Get started Required Technical Background Extension Parts Architecture Overview Development and Testing Pop-Out Extensions Twitch PubSub JSON Web Tokens (JWTs)

Keep Your Secret Private Opaque IDs Focus Designing Extensions Building Extensions Using the Twitch API in an Extension Front End Life Cycle Management Monetization A/B Testing Using Google Analytics Increase Feedback Load Testing Extensions Submission Best Practices Guidelines & Policies Reference Insights & Analytics Mobile Deep Links PubSub Video Broadcast Twitch CLI Changelog Product Lifecycle Technical Background Extension Parts An extension has up to four parts, two required and two optional.

The required parts are: An HTML/JavaScript front end for the broadcaster’s installation configuration.

This is rendered as part of the normal extension installation flow, permitting one-time or infrequent setup.

An HTML/JavaScript front end for the viewer’s experience.

This is rendered in an iframe (inline frame) on the channel page.

The optional parts are: An HTML/JavaScript front end for the broadcaster’s live experience.

['authentic']

This is rendered in an iframe on the broadcaster’s live dashboard page.

It allows the broadcaster to perform privileged operations while live (e.g., creating a new poll).

An Extension Backend Service (EBS), which stores data or state and communicates with the front ends, by receiving AJAX requests or using Twitch’s PubSub architecture to broadcast to all viewers.

Architecture Overview The following figure shows the overall architecture of an Extension.

In the figure, the box labelled “Front-end JavaScript HTML” represents all three front-end parts mentioned above.

Development and Testing An extension’s iframe must import the Extension Helper JavaScript file, created and hosted by Twitch, which provides methods for dealing with authentication, receiving notifications of stream properties, and listening to PubSub events.

Twitch PubSub is a system that allows backend services to broadcast real-time messages to clients.

While an extension is being tested, the iframe and all its assets are sourced from a URL provided by the extension developer, allowing rapid development iteration.

Once the extension is ready for review by Twitch, and later in production, assets to be hosted by Twitch are copied to the Twitch CDN (Content Delivery Network) and served from there.

Pop-Out Extensions If pop-out control is enabled, the extension iframe has an icon button that viewers can click to pop out the extension into its own window.

If an extension is popped out: The extension iframe resizes to take up almost the full size of the new window (all but a minimal, Twitch-controlled UI that wraps the extension).

The channel page retains a placeholder where the popped-out extension was previously.

['authentic']

Twitch Extensions uses JWTs in two contexts: When an extension is loaded in a broadcaster’s or viewer’s browser, a JWT is signed by Twitch and provided to the Extension Helper’s onAuthorize d callback.

Whenever your extension communicates with your EBS, it is best practice to send the JWT and verify it using your extension’s shared secret.

This allows your EBS to verify that the request was made on behalf of the viewer or broadcaster represented by the token, and it prevents malicious users from directly calling your EBS.

For details about the callback, see the Extensions Reference.

When an EBS needs to call a Twitch API, it must generate, sign, and include a JWT in the header.

This is known as “bearer token authentication.”

By signing your own token with the shared secret, Twitch can authenticate that the API request is coming from your EBS.

Note: The secret provided to you in the Extensions console is a base64-encoded string.

Depending on the JWT library you use, you probably will have to decode the string before using it as a parameter to sign or verify a JWT.

Example: Verifying a JWT const secret

= new Buffer('NOTAREALSECRET==', 'base64'); jwt.verify(token, secret);

Example: Signing a JWT const secret

= new Buffer('NOTAREALSECRET==', 'base64')

; const signedToken = jwt.sign(tokenPayload, secret); Keep Your Secret Private While this code contains the secret as an example, we strongly recommend that you do not store your secret in your code.

It is always best practice to store sensitive information such as secrets, keys, and passwords separate from code, using a dedicated secret-management system.

Authentication Tokens

The Extension Helper provides the iframe with an authentication JWT.

When the iframe wants to communicate with its EBS, it sends this token in an HTTP header to the EBS.

The JWT is signed by Twitch, using a secret shared between Twitch and the extension developer.

The EBS can then use this secret to verify the incoming JWT and ensure that received messages are from a legitimate source.

Also, the JWT itself contains reliable information about the role of the sender; e.g., whether the incoming message is from a viewer or a broadcaster.

Opaque IDs Opaque IDs identify viewers without revealing the viewer’s Twitch identity.

Using an opaque ID, developers can determine a viewer’s authenticated state.

Logged-out users also have an opaque ID, but it is not guaranteed to be the same between channels or sessions.

Opaque IDs persist across all channels, and they do not change unless viewers explicitly request a change (rotation).

We encourage developers to use their EBS to store per-viewer information, using opaque IDs as keys.

If your extension needs to know the viewer’s numeric Twitch ID, use the Request Identity Link field on the Extensions Manager Capabilities tab.

Note: An opaque ID that begins with “A” is a logged-out user and should not be persisted.

It will change every time the logged-out user loads your extension.