twitch.json

misinformation

Page ID: 86

Source: https://dev.twitch.tv/docs/extensions/building/

['authentic']

Building Extensions | 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 Designing Extensions Building Extensions Creating An Extension Front End Creating Your Extension Backend Service (EBS) Using the Configuration Service Managing Extension Secrets 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 Building Extensions Creating An Extension Front End As previously defined, extensions are front-end iframes.

Now that you have created your extension and defined its settings, you are ready to create the assets that will live within the iframe.

Mobile Support

If your extension is intended for mobile devices, you can provide a second front end for those devices.

You can use an identical front end for mobile and web, if it is responsive enough to render and perform well on both platforms; otherwise, submit different front ends.

On the Twitch developer site, you are prompted to submit separate front ends for web and mobile.

If you use the same viewer HTML for both web and mobile, we enable you to customize your behavior based on the native platform by providing your viewer with the query strings ?platform=mobile and ?platform=web .

['authentic']

Proper layout, rendering, performance, and interaction on mobile devices are not the same as they are on the web, so if you provide mobile support, test your extension thoroughly on the Twitch mobile app.

Note: Due to requirements communicated to us by Apple, Extensions support on iOS devices (including the ability to test on iOS) is restricted to Apple Developer Program members.

If you develop a mobile Extension for iOS, please make sure you have an Apple developer account, and then fill out this form to request iOS support.

Extension Helper Library

An extension’s iframe must import the Extension Helper JavaScript file, created and hosted by Twitch.

It provides methods for dealing with authentication, receiving notifications of stream properties, and listening to PubSub events.

All HTML files included in your Extension must load the Extension Helper.

To do so, include this line:

<script src="https://extension-files.twitch.tv/helper/v1/twitch-ext.min.js"></script>

For details on the Extension Helper, including the callbacks and functions it provides, see the Extensions Reference .

Creating Your Extension Backend Service (EBS)

The EBS is your optional backend service that supports the extension.

['authentic']

For more information about secrets, see the next section ( Signing the JWT ).

The JWTs used by Twitch Extensions expire, and verification of them fails after the expiration date.

The Extension Helper automatically refreshes the token and then re-calls the onAuthorized() callback.

Always use the latest JWT supplied by the Extension Helper.

For the full JWT schema and detailed notes on each field, see the “ JWT Schema ” section of the Extensions Reference .

Signing the JWT In addition to verifying tokens signed by the Extension Helper, your EBS needs to be able to sign new JWTs for calls to various Extensions endpoints that use JWT as the authentication mechanism.

Use the following format for the payload object in a JWT signed by your EBS: { "exp" : 1502646259, "user\_id" : "27419011" , "role" : "external" } Where: exp is the Unix epoch timestamp when the payload will expire.

Be sure to provide a buffer, to allow potential positive time drift.

user\_id is the Twitch user ID that owns the extension.

role is set to external .

For more information about these fields, see the “JWT Schema” section of the the Extensions Reference .

Sign the payload using your JWT library of choice, which usually has a calling interface similar to this: sign(<payload>, <secret>) Where: <payload> is the token object created in the previous step.

['authentic']

The configuration service enables you to store persistent per-channel and per-extension data and have it provided to your front end on extension startup.

This is a common need for most extensions.

With the configuration service, you can quickly support scenarios like: Enabling broadcasters to customize your extension.

Storing user IDs to call third-party APIs from your EBS.

Note that this information becomes public when it is stored.

All data is sent to extension views by anonymous users, so no authentication is required to retrieve it.

Saving extension-wide settings.

More importantly, with the configuration service, you do not need to expose your EBS to the extension front end on initial load, eliminating the need for your EBS to scale to support that scenario.

In some cases, you can build an extension without an EBS, just by using the configuration service.

(You can still build this functionality in your EBS, if you choose.)

You choose whether to use the configuration service in the capabilities section of extension management on the developer site.

To make your choice, select one of the options (“No configuration”, “Custom/My Own Service”, or “Extension Configuration Service”) on the radio control on that page.

['authentic']

First, set the required versions on the Extensions manager Capabilities tab.

Second, on the extension side, set versions when you set configuration (via the helper method or the API).

Note that these two settings are compared when the channel is loaded.

If they do not match, the extension will not load.

Managing Extension Secrets Each extension maintains a shared secret that is used to sign and verify JSON Web Tokens (JWT) that provide the identity of users.

Use this authentication method when making Extensions API calls from your EBS (for endpoints that support it).

Twitch extension technology relies on a secret shared between the Twitch API and the EBS, to validate JWTs.

This secret has an extremely long life (100 years); however, we strongly recommend that extension developers rotate the shared secret often, to better ensure its security.

JWT Roles Both the EBS and Twitch create JWTs.

The EBS should create and sign JWTs with the external role to perform API actions.

Twitch creates JWTs with other roles, so the EBS can perform user authentication.

Both use cases (the external role and other roles) use the same secret.

(For a discussion of roles, see the “ JWT Schema ” section of the Extensions Reference .)

Creating Secrets To create a new secret: Go to the Settings page of your extension in the Extensions console.

On the left panel, click Secret Keys .

Click Create New Secret to generate a new secret key.

You will see the following: Key — The secret, base64 encoded.