

# oEmbed JS — Complete Plugin Documentation

## 1. What Is oEmbed JS?

oEmbed JS is a lightweight WordPress plugin that automatically converts plain links (e.g., a YouTube URL, a Spotify track link, a tweet) into rich, interactive embeds — directly in the visitor's browser using JavaScript.

### Key design principles:

- **No PHP rendering on the front end.** The plugin does not modify your post content in the database or during page generation. All embed replacement happens client-side, after the page loads.
- **No database tables.** Configuration is stored in a simple JSON file ( `config.json` ) inside the plugin folder.
- **Page-builder friendly.** Instead of hooking into WordPress's content filters, you paste a single `<script>` snippet into your page builder's footer scripts area. This makes it compatible with Bricks Builder, Divi, Elementor, Oxygen, and virtually any builder that allows custom scripts.

## 2. What Is oEmbed?

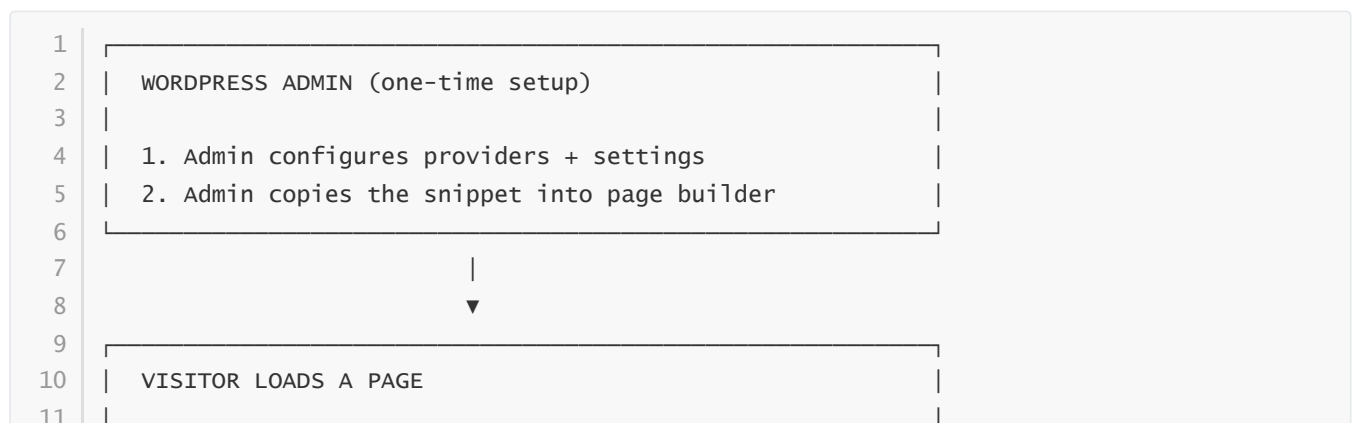
oEmbed is an open standard (oembed.com) that allows websites to provide embed information for their content via a simple API. When you give a provider like YouTube a URL such as `https://www.youtube.com/watch?v=dQw4w9WgXcQ`, their oEmbed endpoint returns a structured JSON response containing:

- **type:** `video`, `rich`, `photo`, or `link`
- **html:** A ready-to-use HTML snippet (usually an `<iframe>` for video, or a `<blockquote>` with a `<script>` for rich embeds like tweets)
- **width** and **height:** Suggested dimensions
- **title**, **author\_name**, **thumbnail\_url**, etc.

oEmbed JS uses this standard to fetch embed code for any supported URL and inject it into your page.

## 3. How the Plugin Works — The Big Picture

Here is the complete flow, from page load to visible embed:



12		3. Page HTML loads, including the <code>&lt;script&gt;</code> snippet	
13		4. Frontend JS ( <code>oembed-js-front.js</code> ) starts	
14		5. JS fetches config from WP REST API ( <code>public-config</code> )	
15		6. JS scans the DOM for links inside target containers	
16		7. For each matching link:	
17		a. Identifies the provider (YouTube, Spotify, etc.)	
18		b. Calls WP REST proxy → proxy calls provider API	
19		c. Receives oEmbed JSON data	
20		d. Replaces the <code>&lt;a&gt;</code> link with the embed HTML	
21		e. If responsive, wraps <code>iframe</code> in aspect-ratio box	
22		f. Executes any <code>&lt;script&gt;</code> tags from the embed HTML	
23			

## 4. Architecture & Components

The plugin consists of these files:

### Core Plugin File — `oembed-js.php`

This is the main PHP file that WordPress loads. It handles:

- **Configuration management:** Reading, writing, and resetting the JSON config via the helper functions `oejs_read_config`, `oejs_write_config`, and `oejs_reset_config`.
- **Admin menu registration:** Adds an „oEmbed JS“ page under WordPress's Settings menu.
- **REST API endpoints:** Registers four endpoints under the `oembed-js/v1` namespace for saving config, resetting config, serving public config, and proxying oEmbed requests.
- **Asset enqueueing:** Loads the admin CSS and JavaScript only on the plugin's settings page.

### Configuration Files

- **defaults.json** — The factory-default configuration. Contains all built-in provider definitions (YouTube, Vimeo, Spotify, Twitter, etc.) and default global settings. This file is never modified by the plugin; it serves as the template for resets.
- **config.json** (created at runtime) — The active configuration. Created by copying `defaults.json` the first time the plugin runs, then updated whenever the admin saves settings. This file lives in the plugin directory and is the single source of truth for all configuration.

### Admin Interface

- **admin/admin-page.php** — The HTML template for the settings page. Contains the form fields, the snippet display area, and an HTML template (inside a `<script type="text/html">` tag) used by JavaScript to render provider rows dynamically.
- **admin/admin.js** — The admin JavaScript that powers the settings page. Handles populating form fields from config, rendering the provider list, adding/deleting providers, collapsing/expanding provider panels, gathering form data, and communicating with the REST API to save or reset.
- **admin/admin.css** — Styling for the admin page (provider panels, snippet box, status messages, etc.).

## Frontend Script

- **frontend/oembed-js-front.js** — The single JavaScript file that runs on your public-facing pages. This is the heart of the plugin for visitors. It fetches the public config, scans the page for matching links, calls the server-side proxy to get oEmbed data, and renders the embeds.

## 5. Installation & Setup

1. **Upload** the `oEmbedJS` folder to `wp-content/plugins/`.
2. **Activate** the plugin in WordPress → Plugins.
3. Navigate to **Settings** → **oEmbed JS**.
4. Review the default settings and providers (all major platforms are pre-configured).
5. **Copy the snippet** from the top of the settings page.
6. **Paste the snippet** into your page builder's footer scripts area.

That's it. Any plain URL link in your content that matches a configured provider will automatically become an embed.


## 6. Configuration — Global Settings

These settings control how all embeds behave by default. Individual providers can override some of these.

Setting	Description	Default	Examples
<b>Maximum Width</b>	The maximum width of the embed wrapper. Accepts any CSS value.	<code>100%</code>	<code>640px</code> , <code>50vw</code> , <code>40rem</code>
<b>Maximum Height</b>	The maximum height. Leave empty for no limit.	<i>(empty)</i>	<code>480px</code> , <code>60vh</code>
<b>Wrapper CSS Class</b>	The CSS class name applied to the <code>&lt;div&gt;</code> that wraps each embed. Use this to target embeds with your own CSS.	<code>jsemb</code>	<code>my-embed</code> , <code>custom-embed</code>
<b>CSS Selectors to Scan</b>	Comma-separated CSS selectors. The frontend script will <b>only</b> look for links inside elements matching these selectors. This is a critical security and performance feature — it prevents the script from processing navigation menus, sidebars, etc.	<code>.entry-content</code> , <code>.post-content</code> , <code>.page-content</code> , <code>.brxe-text</code> , <code>.et_pb_text_inner</code> , <code>article</code>	<code>.my-content-area</code>
<b>Loading Strategy</b>	Controls when embeds load. See below.	<code>lazy</code>	—

Setting	Description	Default	Examples
<b>Responsive Embeds</b>	When enabled, iframes are wrapped in an aspect-ratio container so they scale properly on all screen sizes.	<code>true</code>	—

## Loading Strategies Explained

- **Lazy** (`lazy`): Embeds are fetched immediately, but iframes get the `loading="lazy"` attribute, meaning the browser delays loading the iframe content until the user scrolls near it. Best for performance.
- **Eager** (`eager`): Everything loads immediately when the page loads. Use this if embeds are above the fold and you want them visible instantly.
- **Click to Load** (`click`): A placeholder box with „ Click to load embed“ is shown instead of the embed. The oEmbed API is only called when the user clicks. Best for pages with many embeds or for privacy-conscious sites (no third-party requests until the user opts in).

## 7. Configuration — Providers

Each provider represents a service whose URLs can be embedded. The plugin ships with 15 pre-configured providers:

Provider	Service	Embed Type
YouTube	YouTube videos	iframe
YouTube Shorts	YouTube short-form videos	iframe
Vimeo	Vimeo videos	iframe
Dailymotion	Dailymotion videos	iframe
Spotify	Tracks, albums, playlists, episodes, shows	iframe
SoundCloud	Tracks and sets	rich
Mixcloud	Mixes and shows	rich
X (Twitter)	Tweets	rich
Instagram	Posts, Reels, IGTV	rich
TikTok	TikTok videos	rich
CodePen	Pens	rich
Flickr	Photos	photo
Reddit	Posts and comments	rich
SlideShare	Presentations	rich
Giphy	GIFs	photo

## Provider Fields

Each provider has these configurable fields:

- **Name:** A human-readable label (e.g., „YouTube“). Displayed in the admin panel.
- **URL Pattern:** A regular expression (or plain text prefix) that determines which URLs belong to this provider. For example, YouTube's pattern is:

```
1 | https?://(?:www\.)?(?:youtube\.com/watch?v=|youtu\.be/)([a-zA-Z0-9_-]+)
```

This matches both `youtube.com/watch?v=...` and `youtu.be/...` short links.

- **Endpoint URL:** The oEmbed API endpoint for this provider. For YouTube: `https://www.youtube.com/oembed`.
- **Regex:** Whether the URL Pattern should be treated as a regular expression (checked) or a plain text match (unchecked). Almost all providers use regex.
- **Enabled:** Toggle a provider on or off without deleting it.
- **Embed Type:** How the oEmbed response is typically structured — `iframe` (video/audio players), `rich` (HTML with scripts, like tweets), or `photo` (a simple image).
- **Per-Provider Display Overrides:** You can override Max Width, Max Height, and Wrapper Class for a specific provider. For example, you might want Twitter embeds to be narrower (`max_width: 550px`) while YouTube videos span the full width.

## Adding a Custom Provider

Click „+ Add Provider“ and fill in:

1. A descriptive **Name**.
2. A **URL Pattern** regex that matches the service's URLs.
3. The service's **oEmbed Endpoint URL** (consult the service's API documentation or check `oembed.com/providers.json`).
4. Enable **Regex** if your pattern uses regular expressions.
5. Choose the appropriate **Embed Type**.

---

## 8. The Page Builder Snippet

At the top of the settings page, you'll find a text box containing something like:

```
1 | <script src="https://yoursite.com/wp-content/plugins/oEmbedJS/frontend/oembed-js-  
  | front.js"  
2 |     data-oembed-api="https://yoursite.com/wp-json/oembed-js/v1/public-config"  
3 |     defer></script>
```

**What each part does:**

- `src="...oembed-js-front.js"`: Loads the frontend JavaScript file.

- `data-oembed-api="...public-config"`: Tells the script where to fetch its configuration. This is a public REST API endpoint that returns only the settings and enabled providers — no sensitive information.
- `defer`: Ensures the script doesn't block page rendering. It runs after the HTML is parsed.

#### Where to paste it:

- **Bricks Builder**: Settings → Custom Code → Body (footer) scripts
- **Elementor**: Settings → Custom Code (or via a plugin like „Insert Headers and Footers“)
- **Divi**: Theme Options → Integration → Add code to the body
- **GeneratePress / Astra / etc.**: Use a „custom scripts“ plugin or your theme's footer code area
- **Any theme**: You can also add it to your theme's `footer.php` before `</body>`, or use `wp_footer` hook

**Important:** The snippet goes on the **front end** (public pages), not in the WordPress admin. Some page builders may require you to wrap it differently — consult your builder's documentation.

## 9. How a Link Becomes an Embed — Step by Step

Let's trace what happens when a visitor views a page containing this link:

```
1 <div class="entry-content">
2   <a href="https://www.youtube.com/watch?v=dQw4w9WgXcQ">https://www.youtube.com/watch?
   v=dQw4w9WgXcQ</a>
3 </div>
```

### Step 1: Script Initialization

The `init` function in the frontend script runs. It reads the `data-oembed-api` attribute from its own `<script>` tag to determine the config API URL. It also derives the `proxyurl` by replacing `public-config` with `proxy` in that URL.

### Step 2: Fetch Configuration

The script makes a `GET` request to `/wp-json/oembed-js/v1/public-config`. The server-side `oejs_rest_public_config` function reads the config file and returns only the settings object and the list of **enabled** providers. Disabled providers are filtered out.

### Step 3: Compile Provider Patterns

The `compileProviders` function converts each provider's `url_pattern` string into a JavaScript `RegExp` object for efficient matching. Invalid regex patterns are caught and skipped with a console warning.

### Step 4: Scan the DOM

The `processPage` function uses the configured CSS selectors (e.g., `.entry-content`) to find container elements. It then queries for all `<a href="...">` tags within those containers.

## Step 5: Match a Provider

For each link, `processLink` calls `matchProvider` to test the `href` against all compiled patterns. The YouTube pattern `https://(?:www\.)?(?:youtube\.com/watch?v=|youtu\.be/)([a-zA-Z0-9_-]+)` matches our URL, so the YouTube provider is selected.

## Step 6: Create Placeholder

A `<div class="jsemb">` wrapper is created and inserted into the DOM, replacing the original `<a>` tag. If the loading strategy is „click,“ a clickable placeholder is shown. Otherwise, a „Loading embed...“ message appears temporarily.

## Step 7: Fetch oEmbed Data via Proxy

The script calls `fetchAndEmbed`, which makes a request to:

```
1 | /wp-json/oembed-js/v1/proxy?  
   endpoint=https://www.youtube.com/oembed?url=https://www.youtube.com/watch?v=dQw4w9WgXcQ
```

An in-memory `cache` object prevents duplicate requests — if the same URL appears twice on a page, the second instance uses the cached response.

## Step 8: Server-Side Proxy Fetches from YouTube

The WordPress `oejs_rest_proxy` function:

1. Validates that the requested endpoint ( `https://www.youtube.com/oembed` ) matches one of the configured and enabled providers (prevents abuse as an open proxy).
2. Constructs the full oEmbed API URL: `https://www.youtube.com/oembed?url=https://www.youtube.com/watch?v=dQw4w9WgXcQ&format=json`
3. Fetches it server-side using WordPress's `wp_remote_get()`.
4. Returns the JSON response to the browser with a 1-hour cache header.

## Step 9: Render the Embed

The `renderEmbed` function receives the oEmbed JSON. For YouTube, the response includes an `html` field containing an `<iframe>` tag. The function:

1. Injects the HTML into the wrapper `<div>`.
2. If **responsive** is enabled, finds the `<iframe>`, calculates the aspect ratio from the width/height values (e.g.,  $480/854 = 56.21\%$ ), and wraps it in a `position: relative; padding-bottom: 56.21%` container. The iframe is positioned absolutely to fill this container, ensuring it scales correctly on any screen size.
3. Sets the `loading` attribute (`lazy` or `eager`).
4. Adds `allowfullscreen` and `allow="autoplay; encrypted-media; picture-in-picture"`.

## Step 10: Execute Embed Scripts

For rich embeds (Twitter, Instagram, TikTok), the oEmbed HTML often includes `<script>` tags that need to run to render the embed properly. The script finds these, creates new `<script>` elements (because innerHTML-injected scripts don't execute), and appends them to `document.body`. External scripts (like Twitter's `widgets.js`) are only loaded once even if multiple tweets exist on the page.

---

## 10. The Server-Side Proxy

---

### Why a Proxy?

Many oEmbed providers (YouTube, Vimeo, etc.) don't include CORS headers on their oEmbed API responses. This means a browser-side `fetch()` call directly to `https://www.youtube.com/oembed?url=...` would be blocked by the browser's same-origin policy.

The solution: the browser calls your own WordPress site (`/wp-json/oembed-1.0/proxy`), and your server makes the request to the provider on the browser's behalf. Since server-to-server requests aren't subject to CORS, this works reliably.

### How It Works

The `oejs_rest_proxy` endpoint:

1. Accepts `endpoint` and `url` as query parameters.
2. Validates the endpoint against configured providers.
3. Calls the provider using `wp_remote_get()` with a 15-second timeout.
4. Returns the JSON response with `Cache-Control: public, max-age=3600` (1 hour) so browsers and CDNs can cache it.

### Caching

Caching happens at two levels:

- **Browser-level:** The `Cache-Control` header on proxy responses means repeat visits won't re-fetch the same oEmbed data for an hour.
  - **In-memory (per page load):** The frontend script's `cache` object prevents duplicate requests within a single page view (e.g., if the same YouTube video is linked twice).
- 

## 11. Security & Safety

---

### Admin Protection

- All admin REST API endpoints (`/config`, `/reset`) require `manage_options` capability — only WordPress administrators can save or reset settings.
- The admin JavaScript uses WordPress's nonce system (`X-WP-Nonce` header) to prevent CSRF attacks.



## Proxy Abuse Prevention

The proxy endpoint is public (no authentication required, since visitors need it), but it validates every request against the configured provider list. If someone tries to use the proxy to fetch from an arbitrary URL, the request is rejected with HTTP 403. This prevents your server from being used as an open proxy.

## Content Security

- The plugin uses `escHtml` and `escAttr` functions to sanitize content before inserting it into the DOM.
- Links that fail to embed (provider returns an error, network failure, etc.) are gracefully restored as plain clickable links.
- The configuration file (`config.json`) is stored server-side and never exposed to the browser in full — the public endpoint only returns settings and enabled providers.

## No Database Modifications

The plugin writes nothing to the WordPress database. Your posts, pages, and content remain unchanged. If you deactivate the plugin, embeds simply stop rendering and visitors see the original links.

---

## 12. Troubleshooting

### Embeds aren't appearing

1. **Check the snippet:** Open your page source (View Source) and verify the `<script>` tag with `data-oembed-api` is present in the `<body>`.
2. **Check CSS selectors:** The plugin only processes links inside containers matching your configured selectors. If your theme uses `.my-custom-class` for content areas, add it to the selectors list.
3. **Check browser console:** Open Developer Tools (F12) → Console. Look for warnings starting with „oEmbed JS:“. Common messages:
  - „No data-oembed-api attribute found“ — the snippet wasn't added correctly.
  - „No CSS selectors configured“ — add selectors in the admin.
  - „Failed to load config“ — the REST API might be blocked (check permalink settings).

### A specific provider doesn't work

1. Ensure the provider is **enabled** (checkbox checked).
2. Test the URL pattern: Open your browser console and run:

```
1 | new RegExp("your_pattern_here", "i").test("https://example-url.com/...")
```

3. Some providers (notably **Instagram**) require API tokens. Check the provider's oEmbed documentation.

## REST API returns 404

WordPress REST API requires **pretty permalinks** to be enabled. Go to Settings → Permalinks and choose any option other than „Plain.“

## Embeds load slowly

- Switch loading strategy to **Lazy** so off-screen embeds don't load until needed.
- Consider **Click to Load** for pages with many embeds.
- The 1-hour cache header means return visitors load instantly from browser cache.

---

## 13. FAQ

### Q: Does this plugin work without a page builder?

A: Yes. You can add the snippet to your theme's `footer.php` or use the `wp_footer` action hook. The plugin doesn't depend on any specific page builder.

### Q: Will this conflict with WordPress's built-in oEmbed?

A: Generally no. WordPress's built-in oEmbed operates during content saving/rendering on the server side, converting URLs in the block editor. oEmbed JS operates entirely client-side on plain `<a>` links. They target different things. If you see double embeds, either disable the built-in one for specific post types or adjust your CSS selectors to avoid overlap.

### Q: What happens if I deactivate the plugin?

A: Visitors will simply see the original `<a>` links. Nothing in your database or content is changed. Reactivate and embeds return.

### Q: Can I style the embeds?

A: Yes. Target the wrapper class (default `jsemb`) in your CSS:

```
1 .jsemb {  
2   margin: 20px 0;  
3   border-radius: 8px;  
4   overflow: hidden;  
5   box-shadow: 0 2px 8px rgba(0,0,0,0.1);  
6 }
```

### Q: Can I add a provider that isn't in the default list?

A: Absolutely. Click „+ Add Provider“ in the admin, fill in the URL pattern and oEmbed endpoint for any service that supports the oEmbed standard. Check <https://oembed.com/providers.json> for a comprehensive list.

### Q: Is the configuration backed up?

A: The config is stored in `config.json` in the plugin folder. It's not in the database, so standard database backups won't include it. Include the plugin folder in your file-level backups, or export the JSON manually.

### Q: Why use a JSON file instead of the WordPress database?

A: By design, to keep the plugin lightweight and database-free. The config is small (a few KB), rarely changes, and is read more often than written — a flat file is perfectly suited. It also makes the configuration trivially portable: copy the `config.json` file to another site and you're done.