WP Plugins by AI

AI generated WP plugins (Opus + ChatGPT)

- oEmbed JS
  - oEmbed JS (v1.2.0) — Documentation
  - Skill: oEmbed JS Provider Configuration Assistant

oEmbed JS

A plugin that lets you define your own oEmbeds for WordPress. It also works in page builders like BricksBuilder and Divi.

oEmbed JS (v1.2.0) — Documentation

> Update 1.3.0: You can now change the order of providers and disable WordPress's built-in oEmbed.

Introduction

oEmbed JS is a WordPress plugin that transforms plain URLs in your content into rich embedded media — videos, audio players, social media posts, images, and more — entirely using client-side JavaScript. Unlike WordPress's built-in oEmbed system, which renders embeds during page generation on the server, oEmbed JS operates in the browser. This makes it particularly well-suited for sites built with page builders like Bricks Builder, Elementor, Divi, and others where you may not have direct control over how post content is rendered by PHP.

The plugin ships with 15 pre-configured providers including YouTube, Vimeo, Spotify, SoundCloud, X (Twitter), TikTok, and more. You can add any oEmbed-compatible provider manually or import from the official oEmbed directory, which contains over 300 providers.

What is oEmbed?

oEmbed is an open standard that allows websites to display embedded representations of a URL. Instead of requiring you to copy and paste complex `<iframe>` code, oEmbed lets a consuming site (yours) ask a provider site (like YouTube) for the embed code automatically.

The protocol works through a simple request-response cycle:

1. **Discovery**: Your site recognizes that a URL belongs to a specific provider (e.g., `https://www.youtube.com/watch?v=dQw4w9WgXcQ` belongs to YouTube).

2. **Request**: Your site sends a request to the provider's oEmbed endpoint — a special API URL — asking "How should I display this URL?" The request looks something like:

```
https://www.youtube.com/oembed?url=https://www.youtube.com/watch?v=dQw4w9WgXcQ&format=json
```

3. **Response**: The provider returns a JSON object containing metadata and embed code:

```
{
  "type": "video",
  "title": "Rick Astley - Never Gonna Give You Up",
  "html": "<iframe width=\"480\" height=\"270\" src=\"https://www.youtube.com/embed/dQw4w9WgXcQ\" ...></iframe>",
  "width": 480,
  "height": 270,
  "provider_name": "YouTube"
}
```

4. **Rendering**: Your site takes the `html` field from the response and inserts it into the page, replacing the original link.

The oEmbed response `type` field can be one of four values:

- **video** — Typically returns an `<iframe>` embed for video or audio players.
- **rich** — Returns arbitrary HTML, used by services like Twitter, Instagram, and TikTok for their post embeds.
- **photo** — Returns a direct image URL rather than HTML.
- **link** — A simple hyperlink with title metadata; no visual embed.

Why a Proxy is Needed

Modern browsers enforce a security policy called **CORS** (Cross-Origin Resource Sharing). When JavaScript running on your site tries to fetch data from a different domain (like `youtube.com/oembed`), the browser blocks the response unless the remote server explicitly allows it with CORS headers. Many oEmbed providers do not include these headers.

oEmbed JS solves this by routing oEmbed requests through your own WordPress server. The frontend JavaScript sends its request to a proxy endpoint on your site (`/wp-json/oembed-js/v1/proxy`), which then fetches the data from the provider server-to-server, where CORS restrictions do not apply, and passes the response back to the browser.

How oEmbed JS Works

The plugin operates in two distinct phases:

Phase 1: Configuration (Admin)

When you visit **Settings → oEmbed JS** in your WordPress admin, the plugin reads its configuration from a JSON file stored in the plugin directory. The admin page is a single-page application that communicates with the WordPress REST API to save settings, reset defaults, and browse the oEmbed directory. No settings are stored in the WordPress database — everything lives in a `config.json` file.

Phase 2: Embedding (Frontend)

When a visitor loads a page on your site, the frontend script (included via the snippet you paste into your page builder) performs the following sequence:

1. **Fetches configuration** from the public REST endpoint (`/wp-json/oembed-js/v1/public-config`), which returns only the enabled providers and display settings.
2. **Compiles URL patterns** from the provider list into regular expressions.
3. **Scans the DOM** for `<a>` (anchor) elements within the CSS selectors you have configured (e.g., `.entry-content`, `.brxe-text`).
4. **Matches each link** against the compiled provider patterns to determine if it should be embedded.
5. **Replaces matching links** with embed wrappers and fetches the oEmbed data through the server-side proxy.
6. **Renders the embed** HTML returned by the provider, applying your display settings (responsive wrapping, dimensions, lazy loading, etc.).
7. **Executes embedded scripts** if the provider's HTML includes `<script>` tags (required by Twitter, Instagram, TikTok, and similar services that use JavaScript-based embed rendering).

Installation

1. Upload the `oembedjs_1.2.0` folder to your `/wp-content/plugins/` directory, or install through the WordPress plugin installer.
2. Activate the plugin through the **Plugins** menu in WordPress.
3. Navigate to **Settings → oEmbed JS** to configure the plugin.

4. Copy the generated snippet and paste it into your page builder's footer scripts section.

Requirements

- WordPress 5.0 or higher
- PHP 7.0 or higher
- The plugin directory must be writable by the web server (for storing `config.json`)

Getting Started

After activation, the basic setup requires two steps:

1. **Define CSS Selectors**: Tell the plugin which parts of your pages contain links that should be converted to embeds. This is done in the "CSS Selectors to Scan" field under Global Display Settings.
2. **Install the Snippet**: Copy the snippet from the top of the settings page and paste it into your page builder's footer scripts area.

Once these two steps are complete, any plain link in your content that matches a configured provider will automatically be replaced with a rich embed when visitors view the page.

The Snippet

At the top of the settings page, you will find a **Page Builder Snippet** box containing a `<script>` tag. This is the frontend loader that makes embedding work. It looks like this:

```
<script src="https://yoursite.com/wp-content/plugins/oembedjs_1.2.0/frontend/oembed-js-front.js" data-oembed-api="https://yoursite.com/wp-json/oembed-js/v1/public-config" defer></script>
```

How the Snippet Works

- The `src` attribute points to the frontend JavaScript file that performs all the link scanning and embedding logic.
- The `data-oembed-api` attribute tells the script where to fetch the plugin's configuration. The script reads this attribute from its own `<script>` tag at runtime.

- The `defer` attribute ensures the script loads without blocking page rendering and executes after the HTML document has been parsed.

Where to Place It

Place this snippet in your page builder's **footer scripts** section. The exact location depends on your builder:

- **Bricks Builder**: Settings → Custom Code → Body (footer) scripts
- **Elementor**: Settings → Custom Code (or use the wp_footer hook)
- **Divi**: Theme Options → Integration → Add code to the body
- **GeneratePress / Astra / other themes**: Use a child theme's `footer.php` or a custom code snippets plugin

The snippet does not need to be wrapped in additional `<script>` tags — it is already a complete script tag.

Use the **Copy Snippet** button to copy the text to your clipboard.

Global Display Settings

These settings control the default appearance and behavior of all embeds. Individual providers can override some of these values.

Maximum Width

Sets the CSS `max-width` property on embed wrappers. Accepts any valid CSS length value.

- **Default**: `100%`
- **Examples**: `100%`, `640px`, `50vw`, `40rem`

This constrains the embed to never exceed the specified width while allowing it to be smaller when the container is narrower.

Maximum Height

Sets the CSS `max-height` property on embed wrappers. Content exceeding this height will be clipped.

- **Default**: Empty (unlimited)

- **Examples**: `480px`, `60vh`

Leave empty to allow embeds to take their natural height.

Fixed Width

When set, applies a CSS `width` (not `max-width`) to the embed wrapper. This overrides the Maximum Width setting.

- **Default**: Empty (auto)
- **Examples**: `640px`, `50vw`

Use this when you want all embeds to be exactly a specific width regardless of their container.

Fixed Height

When set, applies a CSS `height` (not `max-height`) to the embed wrapper. This overrides the Maximum Height setting.

- **Default**: Empty (auto)
- **Examples**: `480px`, `360px`

Important: When fixed dimensions are set, the responsive aspect-ratio wrapping is disabled because you are explicitly controlling the size.

Wrapper CSS Class

The CSS class applied to the `<div>` element that wraps each embed.

- **Default**: `jsemb`

You can target this class in your theme's CSS to style all embeds globally. For example:

```
.jsemb {
  margin: 20px auto;
  border-radius: 8px;
  overflow: hidden;
}
```

CSS Selectors to Scan

A comma-separated list of CSS selectors. The frontend script will only scan for links inside elements matching these selectors.

- **Default**: `.entry-content, .post-content, .page-content, .brxe-text, .et_pb_text_inner, article`

This is a critical security and performance measure. By restricting scanning to specific containers, the plugin avoids converting links in navigation menus, sidebars, footers, or other areas where you would not want embeds to appear.

If this field is empty, no embeds will be generated. This is by design — the plugin requires you to explicitly define where embeds should appear.

To find the correct selectors for your theme, use your browser's developer tools (right-click an element → Inspect) to identify the class or ID of the container that holds your post or page content.

Loading Strategy

Controls when embed content (particularly iframes) is loaded:

- **Lazy** (default): Uses the browser's native `loading="lazy"` attribute on iframes. The embed content is only fetched when the user scrolls it into or near the viewport. This improves initial page load performance.
- **Eager**: Loads all embed content immediately when the page loads. Use this if embeds appear above the fold and you want them visible as quickly as possible.
- **Click to load**: Displays a placeholder box with the provider name and URL. The actual embed is only fetched and rendered when the user clicks the placeholder. This is the most performance-friendly option and also provides a measure of privacy, since no third-party content is loaded until the user explicitly requests it.

Responsive Embeds

When enabled (default), the plugin wraps iframes in an aspect-ratio-preserving container using the "padding-bottom percentage" technique. This ensures video embeds scale fluidly to fill their container width while maintaining the correct height ratio (typically 16:9).

The technique works as follows:

1. The plugin reads the `width` and `height` attributes from the iframe (or from the oEmbed response data).
2. It calculates the aspect ratio: `(height / width) × 100` to get a percentage.
3. It creates a wrapper `<div>` with `position: relative`, `height: 0`, and `padding-bottom` set to that percentage.
4. The iframe is positioned absolutely within this wrapper to fill it completely.

This approach is widely used across the web for responsive video embeds and works in all modern browsers.

When fixed width or fixed height values are set, responsive wrapping is automatically disabled because explicitly sized embeds should not scale.

Providers

Providers are the services whose URLs the plugin recognizes and converts to embeds. Each provider entry has the following properties:

Name

A human-readable label displayed in the admin interface. This has no effect on functionality.

URL Pattern

A text pattern used to match URLs in your content against this provider. This can be either a regular expression or a plain text substring, controlled by the Regex checkbox.

As a regular expression (Regex checked):

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

This uses standard JavaScript regex syntax. The `i` (case-insensitive) flag is automatically applied. The pattern is tested against the full `href` value of each anchor tag.

As a plain pattern (Regex unchecked):

```
youtube.com/watch
```

The plugin checks if the URL contains this substring anywhere within it.

Regular expressions are more precise and prevent false matches. All pre-configured providers use regex patterns.

Endpoint URL

The oEmbed API URL for the provider. This is where the plugin sends requests (via the server-side proxy) to retrieve embed data.

For example, YouTube's endpoint is `https://www.youtube.com/oembed`. When the plugin encounters a matching YouTube URL, it constructs a request like:

```
https://www.youtube.com/oembed?url=https://www.youtube.com/watch?v=dQw4w9WgXcQ&format=json
```

Regex

A checkbox indicating whether the URL Pattern should be treated as a JavaScript regular expression (`true`) or a plain substring match (`false`).

Enabled

A checkbox to quickly enable or disable a provider without deleting it. Disabled providers are excluded from the public configuration served to the frontend and are not checked by the proxy's allowlist validation.

Embed Type

A classification hint for the provider:

- **iframe (video/audio)**: The provider typically returns an `<iframe>` element. Used for YouTube, Vimeo, Spotify, etc.
- **Rich HTML**: The provider returns arbitrary HTML that may include `<script>` tags. Used for Twitter, Instagram, TikTok, Reddit, etc.
- **Photo**: The provider returns an image URL rather than HTML.

This field is informational and used by the admin interface. The frontend rendering logic primarily checks for the presence of `html` in the oEmbed response, the `type` field, and the `url` field for photos, rather than relying on this admin-side classification.

Managing Providers

- **Add Provider**: Click the "+ Add Provider" button to create a blank provider entry. Fill in the name, URL pattern, endpoint, and other fields.
- **Delete Provider**: Click the × button on any provider's header row. You will be asked to confirm.
- **Reorder**: Providers are matched in order from top to bottom. The first matching provider wins. Currently, reordering requires deleting and re-adding providers.

- **Collapse/Expand All**: Utility buttons to collapse or expand all provider panels at once for easier navigation.

Remember: After making any changes to providers or settings, you must click **Save Settings** at the bottom of the page.

The oEmbed Directory Browser

The plugin includes a built-in browser for the official oEmbed provider directory hosted at oembed.com. This directory contains over 300 providers maintained by the oEmbed community.

Accessing the Directory

Click the ◫ **Browse oEmbed Directory** button in the Providers section. A modal window opens with a searchable list of all available providers.

How It Works

1. When you first open the directory, the plugin fetches the provider list from `https://oembed.com/providers.json` through your WordPress server (via the REST endpoint `/wp-json/oembed-js/v1/oembed-directory`).
2. The server transforms each entry from the official format into the plugin's internal format. This transformation includes:
   - Converting oEmbed wildcard scheme patterns (which use `*` as a wildcard) into regular expressions (replacing `*` with `.+`).
   - Normalizing endpoint URLs by replacing `{format}` placeholders with `json`.
   - Generating stable IDs from provider names.
3. The transformed data is cached as a WordPress transient for 24 hours, so subsequent opens of the directory do not re-fetch from oembed.com.
4. The directory list is displayed in the browser. Providers that you have already added (matched by endpoint URL) are visually dimmed and labeled "Already added." By default, already-added providers are hidden; uncheck the "Hide already added" checkbox to show them.

Importing Providers

1. Search or scroll through the list to find providers you want.
2. Click on a provider row (or its checkbox) to select it. You can select multiple providers.
3. Click **Import Selected** to add the selected providers to your configuration.

4. The modal closes and you will see the imported providers in your provider list.
5. **Click Save Settings** to persist the changes.

Imported providers are pre-configured with the URL patterns and endpoints from the official directory. You may need to adjust patterns or settings for specific use cases.

Per-Provider Display Overrides

Each provider has an "overrides" section that lets you customize display settings for that specific provider, overriding the global values. Any override field left empty falls back to the corresponding global setting.

Available overrides:

- **Max Width**: Override the global maximum width for this provider's embeds.
- **Max Height**: Override the global maximum height.
- **Fixed Width**: Override the global fixed width.
- **Fixed Height**: Override the global fixed height.
- **Wrapper Class**: Use a different CSS class for this provider's embed wrappers.

This is useful when different content types need different treatment. For example, you might want Twitter embeds to be narrower than YouTube videos:

- Global Max Width: `100%`
- Twitter override Max Width: `550px`
- Spotify override Fixed Height: `352px`

How the Frontend Script Works

The frontend script at `frontend/oembed-js-front.js` is the core of the plugin's client-side functionality. Here is a detailed walkthrough of its operation:

1. Self-Discovery

The script finds its own `<script>` tag in the DOM by querying for `script[data-oembed-api]`. It reads the `data-oembed-api` attribute to determine the URL of the configuration endpoint. It also derives the proxy URL by replacing `public-config` with `proxy` in the API URL path.

2. Configuration Fetch

The script makes a `fetch()` request to the public configuration endpoint. This endpoint returns only the data the frontend needs: the display settings and the list of **enabled** providers. Disabled providers are filtered out on the server side.

3. Pattern Compilation

Each provider's URL pattern is compiled into a JavaScript `RegExp` object (if it is a regex pattern) or stored as a plain string for substring matching. Invalid regex patterns are caught and logged as warnings without crashing the script. This compilation happens once and the compiled patterns are reused for all URL checks.

4. DOM Scanning

The script queries the DOM using the CSS selectors defined in your settings. Within each matched container, it finds all `<a>` elements with an `href` attribute. Each anchor is processed exactly once — a `data-oejs-processed` attribute is set on each processed link to prevent duplicate handling.

5. URL Matching

For each anchor, the script tests the `href` against all compiled provider patterns in order. The first match wins, and the corresponding provider configuration is used.

6. Wrapper Creation

A `<div>` element is created with the configured wrapper class and dimension styles. This wrapper replaces the original `<a>` element in the DOM. Depending on the loading strategy:

- **Lazy/Eager**: A "Loading embed…" placeholder is shown, and the oEmbed fetch begins immediately.
- **Click to load**: A styled placeholder with the provider name and URL is shown. The oEmbed fetch only begins when the user clicks it.

7. oEmbed Fetching

The script requests embed data through the server-side proxy:

```
/wp-json/oembed-js/v1/proxy?endpoint=<provider_endpoint>&url=<page_url>
```

Responses are cached in an in-memory JavaScript object keyed by the full proxy URL. If the same URL appears multiple times on a page, only one network request is made.

8. Rendering

Based on the oEmbed response:

- If `html` is present (video/rich types), the HTML is inserted directly.
- If the type is `photo` and a `url` field is present, an `<img>` tag is created.
- Otherwise, a fallback link is displayed.

After HTML insertion:

- **Responsive wrapping**: If enabled and no fixed dimensions are set, any `<iframe>` found in the embed HTML is wrapped in an aspect-ratio container.
- **Loading attribute**: The `loading` attribute is set on iframes to `lazy` or `eager` per your configuration.
- **iframe permissions**: The `allowfullscreen` and `allow` attributes are set to enable fullscreen, autoplay, encrypted media, and picture-in-picture.

9. Script Execution

Some providers (Twitter, Instagram, TikTok) return HTML that includes `<script>` tags. These scripts are responsible for rendering the embed's visual appearance. Since scripts inserted via `innerHTML` do not execute automatically in browsers, the plugin explicitly re-creates each `<script>` element:

- For external scripts (`src` attribute): A new `<script>` tag with the same `src` is appended to `<body>`, but only if a script with that `src` does not already exist (avoiding duplicate loading).
- For inline scripts: A new `<script>` tag with the same `textContent` is appended to `<body>`.

The original `<script>` elements are removed from the embed wrapper.

The Server-Side Proxy

The proxy endpoint at `/wp-json/oembed-js/v1/proxy` is a critical component that bridges the gap between browser security restrictions and oEmbed providers.

Request Flow

```
Browser → Your WordPress Server (/wp-json/oembed-js/v1/proxy) → Provider's oEmbed Endpoint → Response flows back
```

Security: Allowlist Validation

To prevent your server from being used as an open proxy (which could be abused to make arbitrary web requests through your server), the proxy validates every request against your configured providers. Specifically:

1. The proxy receives `endpoint` and `url` parameters.
2. It reads the current plugin configuration.
3. It compares the requested `endpoint` against every enabled provider's endpoint URL (normalized: trailing slashes removed, case-insensitive comparison, query parameters stripped before comparison).
4. If no match is found, the request is rejected with a `403 Forbidden` error.

This means only endpoints that you have explicitly configured and enabled in the plugin can be accessed through the proxy.

Caching

The proxy sets a `Cache-Control: public, max-age=3600` response header, instructing browsers and intermediate caches to cache the oEmbed response for one hour. This reduces the number of requests to provider endpoints for frequently viewed pages.

Error Handling

- If the provider returns a non-2xx HTTP status, the proxy returns the same status code with an error message.
- If the provider returns invalid JSON, a `502 Bad Gateway` error is returned.
- If the WordPress HTTP request fails (network error, timeout), a `502` error is returned with the error message.

The proxy uses a 15-second timeout for requests to provider endpoints.

Configuration Storage

oEmbed JS stores all configuration in a single file: `config.json`, located in the plugin's root directory alongside `defaults.json`.

Why a JSON File Instead of the Database?

This design choice has several implications:

- **Portability**: The configuration can be easily backed up, version-controlled, or transferred between environments by copying a single file.
- **Performance**: Reading a small JSON file from disk is fast and does not require a database query.
- **Simplicity**: No database tables need to be created or migrated.

File Permissions

The web server process (typically running as `www-data`, `apache`, or `nginx`) must have write permission to the plugin directory to create and update `config.json`. If the file cannot be written, save operations will fail with an error message indicating the file path.

How Configuration Flows

1. **Admin page load**: PHP reads `config.json` and passes it to the JavaScript admin interface via `wp_localize_script()` as the `OEJS.config` object.
2. **Admin save**: The JavaScript gathers all form values, constructs a JSON object, and POSTs it to `/wp-json/oembed-js/v1/config`. The PHP handler writes the data to `config.json` and returns the saved configuration for verification.
3. **Frontend load**: The frontend script fetches `/wp-json/oembed-js/v1/public-config`, which reads `config.json` and returns a filtered version containing only enabled providers.

defaults.json

The `defaults.json` file is a read-only template containing the factory default configuration. It is used when:

- The plugin is activated for the first time and no `config.json` exists.
- You click "Reset to Defaults" in the admin interface.
- The `config.json` file becomes corrupted or unreadable.

This file should never be edited. It ships with the plugin and is overwritten on updates.

Resetting to Defaults

Clicking **Reset to Defaults** in the admin interface:

1. Displays a confirmation dialog.
2. Sends a POST request to `/wp-json/oembed-js/v1/reset`.
3. The server copies the contents of `defaults.json` over `config.json`.
4. The admin page re-populates all fields with the default values.

This action cannot be undone. Any custom providers or settings modifications will be lost.

Uninstallation

When you delete the plugin through the WordPress admin (not just deactivate — actually delete):

1. WordPress executes the `uninstall.php` file.
2. The plugin deletes the `oejs_oembed_directory` transient from the WordPress database (the cached oEmbed directory data).
3. The plugin's files, including `config.json`, are removed by WordPress's standard plugin deletion process.

Since the plugin stores no other data in the WordPress database (no options, no custom tables), uninstallation is clean and complete.

Note: Deactivating the plugin (without deleting) preserves all files including your `config.json`, so reactivating will restore your settings.

Troubleshooting

Embeds are not appearing

1. **Check CSS Selectors**: The most common issue. If the "CSS Selectors to Scan" field is empty, no links will be processed. Open your browser's developer tools, inspect the element containing your links, and note the CSS class of the content container. Enter it in the selectors field.
2. **Verify the snippet is installed**: View your page source (Ctrl+U in most browsers) and search for `oembed-js-front.js`. If it is not present, the snippet has not been correctly

added to your page builder.
3. **Check the browser console**: Open the developer tools console (F12 → Console tab). The plugin logs informational messages:
   - `"oEmbed JS: No CSS selectors configured."` — The selectors field is empty.
   - `"oEmbed JS: Failed to load config."` — The public-config endpoint is unreachable.
   - `"oEmbed JS: Invalid regex for ..."` — A provider has a malformed URL pattern.
   - `"oEmbed JS: Failed to fetch embed for ..."` — The oEmbed request failed.
4. **Check that the link is a plain anchor**: The plugin only processes `<a>` elements with an `href` attribute. If your page builder wraps URLs in other elements or the links are inside iframes, they will not be detected.
5. **Check provider is enabled**: In the admin, verify the checkbox next to the provider name is checked.

"Error saving settings" message

The web server cannot write to the `config.json` file. Check that the plugin directory has appropriate write permissions. On most Linux servers:

```
chown -R www-data:www-data /path/to/wp-content/plugins/oembedjs_1.2.0/
chmod 755 /path/to/wp-content/plugins/oembedjs_1.2.0/
chmod 644 /path/to/wp-content/plugins/oembedjs_1.2.0/config.json
```

Instagram embeds require a Facebook App token

Instagram's oEmbed endpoint (`graph.facebook.com/v18.0/instagram_oembed`) requires an `access_token` parameter linked to a Facebook App. The pre-configured Instagram provider will not work without this token. You need to create a Facebook App, obtain an access token, and append it to the endpoint URL:

```
https://graph.facebook.com/v18.0/instagram_oembed?access_token=YOUR_TOKEN
```

Embeds appear but look broken or unstyled

Some providers (Twitter, Instagram, TikTok) rely on their own JavaScript libraries to render embeds. If your site has a Content Security Policy (CSP) that blocks scripts from third-party domains, these embeds will not render correctly. Check your browser console for CSP violation warnings.

Embeds load slowly

- Switch the loading strategy to **Lazy** so embeds below the fold are not loaded until needed.
- Consider using **Click to load** for pages with many embeds.
- The server-side proxy caches responses for 1 hour via Cache-Control headers. Your server or a CDN may provide additional caching.

Technical Reference

REST API Endpoints

All endpoints are registered under the namespace `oembed-js/v1`.

Endpoint	Method	Auth Required	Description
`/config`	POST	Yes (`manage_options`)	Save the full plugin configuration
`/reset`	POST	Yes (`manage_options`)	Reset configuration to defaults
`/public-config`	GET	No	Retrieve display settings and enabled providers
`/proxy`	GET	No	Proxy oEmbed requests to provider endpoints
`/oembed-directory`	GET	Yes (`manage_options`)	Fetch and transform the oembed.com directory

File Structure

```
oembedjs_1.2.0/
├── oembed-js.php          – Main plugin file, PHP logic, REST endpoints
├── defaults.json          – Factory default configuration (read-only)
├── config.json            – Active configuration (created at runtime)
├── providers.json         – Local copy of the oembed.com directory
├── uninstall.php          – Cleanup on plugin deletion
├── admin/
│   ├── admin-page.php     – Admin page HTML template
```

```
│   ├── admin.js           — Admin interface JavaScript
│   └── admin.css          — Admin interface styles
└── frontend/
    └── oembed-js-front.js — Frontend embedding script
```

Configuration Schema

The `config.json` (and `defaults.json`) file follows this structure:

```
{
  "settings": {
    "max_width": "100%",
    "max_height": "",
    "fixed_width": "",
    "fixed_height": "",
    "wrapper_class": "jsemb",
    "css_selectors": ".entry-content, .post-content",
    "loading_strategy": "lazy",
    "responsive": true
  },
  "providers": [
    {
      "id": "youtube",
      "name": "YouTube",
      "url_pattern": "https?://(?:www\\.)?youtube\\.com/watch\\?v=...",
      "endpoint": "https://www.youtube.com/oembed",
      "regex": true,
      "enabled": true,
      "embed_type": "iframe",
      "override": {
        "max_width": "",
        "max_height": "",
        "fixed_width": "",
        "fixed_height": "",
        "wrapper_class": ""
      }
    }
  ]
}
```

Pre-Configured Providers

The plugin ships with the following 15 providers enabled by default:

Provider	Endpoint	Embed Type
YouTube	youtube.com/oembed	iframe
YouTube Shorts	youtube.com/oembed	iframe
Vimeo	vimeo.com/api/oembed.json	iframe
Dailymotion	dailymotion.com/services/oembed	iframe
Spotify	open.spotify.com/oembed	iframe
SoundCloud	soundcloud.com/oembed	rich
Mixcloud	app.mixcloud.com/oembed/	rich
X (Twitter)	publish.twitter.com/oembed	rich
Instagram	graph.facebook.com/v18.0/instagram_oembed	rich
TikTok	tiktok.com/oembed	rich
CodePen	codepen.io/api/oembed	rich
Flickr	flickr.com/services/oembed/	photo
Reddit	reddit.com/oembed	rich
SlideShare	slideshare.net/api/oembed/2	rich
Giphy	giphy.com/services/oembed	photo

WordPress Hooks

The plugin uses the following WordPress hooks:

- `admin_menu` — Registers the settings page under the Settings menu.
- `admin_enqueue_scripts` — Loads admin CSS and JavaScript only on the plugin's settings page.
- `rest_api_init` — Registers all five REST API endpoints.

Browser Compatibility

The frontend script uses `fetch()`, `document.querySelectorAll()`, `Element.closest()`, and `classList`. These are supported in all modern browsers (Chrome, Firefox, Safari, Edge). Internet Explorer is not supported.

Skill: oEmbed JS Provider Configuration Assistant

Purpose: When a user wants to add a new oEmbed provider to the oEmbed JS WordPress plugin, help them determine the correct values for each field based on whatever information they provide (a URL, an iframe embed code, a provider website, an API documentation link, or just a service name).

Context: The oEmbed JS plugin allows users to configure custom oEmbed providers. Each provider has the following fields that need to be filled in:

Field	Description	Expected Format	Required
Name	Human-readable name of the provider (e.g., "YouTube", "Spotify")	Plain text	Yes
URL Pattern	A regex pattern that matches URLs from this provider that should be converted to embeds	Regular expression (without delimiters), case-insensitive matching is applied automatically	Yes
Endpoint URL	The oEmbed API endpoint URL where the plugin sends requests to retrieve embed data	A full HTTPS URL, must return JSON	Yes
Regex	Whether the URL Pattern field is a regular expression	Checkbox (true/false). Almost always true.	Yes
Enabled	Whether this provider is active	Checkbox (true/false)	Yes
Embed Type	The type of embed this provider returns	One of: "iframe" (for video/audio players), "rich" (for HTML embeds like tweets, posts), or "photo" (for images)	Yes
Max Width (override)	Per-provider maximum width override	CSS value (e.g., "640px", "100%") or leave empty to use global setting	No
Max Height (override)	Per-provider maximum height override	CSS value (e.g., "480px") or leave empty to use global setting	No

Field	Description	Expected Format	Required
Fixed Width (override)	Per-provider fixed width override	CSS value or leave empty	No
Fixed Height (override)	Per-provider fixed height override	CSS value or leave empty	No
Wrapper Class (override)	Per-provider CSS class override for the embed wrapper	CSS class name or leave empty to use global setting	No

How to derive the values from user input:

1. **If the user provides a URL they want to embed (e.g., "https://www.example.com/video/12345"):**
   - Identify the provider/service from the domain.
   - Search for whether this provider has a known oEmbed endpoint. Check oembed.com/providers.json or the provider's documentation.
   - Construct a regex URL pattern that matches all embeddable URLs from this provider. Use capturing groups and character classes as needed. Escape dots with `\\.` in the regex.
   - If no oEmbed endpoint exists, inform the user that this provider does not support oEmbed and suggest alternatives (e.g., using an iframe embed directly via a custom solution, or checking if the provider offers oEmbed through a third-party service).
2. **If the user provides an iframe embed code (e.g., `<iframe src="https://player.example.com/embed/12345">`)**:
   - Extract the iframe src URL to understand the embed URL structure.
   - Work backwards from the embed/player URL to identify the provider.
   - Look up whether the provider has an oEmbed endpoint.
   - Note: The oEmbed endpoint URL is NOT the iframe src. The endpoint is an API URL that accepts a content URL and returns embed HTML (which typically contains the iframe). These are different things.
   - If the provider has an oEmbed API, determine the correct content URL pattern (the URL a user would share, not the embed URL) and the API endpoint.
3. **If the user provides just a service name (e.g., "Vimeo"):**
   - Look up the provider in the known oEmbed providers directory.
   - Provide the standard configuration.
4. **If the user provides a provider's API documentation link:**
   - Read the documentation to extract the oEmbed endpoint URL and supported URL schemes.
   - Convert the URL schemes to regex patterns.

Converting oEmbed scheme patterns to regex:

Many providers document their supported URLs using wildcard patterns like `https://www.example.com/video/*`. Convert these to regex:

- Replace `*` with `[a-zA-Z0-9_-]+` for path segments, or `.+` for broader matching.

- Escape literal dots: `.` becomes `\\.`
- Use `https?://` to match both HTTP and HTTPS if both are supported.
- Use `(?:www\\.)?` to optionally match the www subdomain if appropriate.
- Combine multiple scheme patterns using `|` (alternation) within a non-capturing group `(?:...|...)` if needed, or use a single broader pattern if all schemes can be covered.

Choosing the Embed Type:

- Use **"iframe"** for video and audio players (YouTube, Vimeo, Spotify, Dailymotion, etc.) — services whose oEmbed response contains an `<iframe>` in the `html` field.
- Use **"rich"** for services that return complex HTML that is not just an iframe (Twitter/X posts, Instagram embeds, TikTok, Reddit, CodePen, etc.) — these often include `<blockquote>` elements with `<script>` tags.
- Use **"photo"** for services that return image data (Flickr photos, Giphy, etc.) — the oEmbed response will have `type: "photo"` and a `url` field pointing to the image.

Response format: Present the recommended configuration in a clear table. Example:

For a user who says: "I want to embed Vimeo videos"

Field	Value
Name	Vimeo
URL Pattern	`https?://(?:www\.)?vimeo\.com/(\d+)`
Endpoint URL	`https://vimeo.com/api/oembed.json`
Regex	☑ Checked
Enabled	☑ Checked
Embed Type	iframe
Max Width	*(leave empty — uses global setting)*
Max Height	*(leave empty — uses global setting)*
Fixed Width	*(leave empty)*
Fixed Height	*(leave empty)*
Wrapper Class	*(leave empty — uses global setting)*

Additional guidance to provide the user:

- After adding the provider, remind the user to click **"Save Settings"** in the plugin admin page.
- If the provider requires authentication (e.g., Instagram requires a Facebook App access token), warn the user and explain that the oEmbed endpoint may need an `access_token` parameter appended. The endpoint field can include query parameters, e.g., `https://graph.facebook.com/v18.0/instagram_oembed?access_token=YOUR_TOKEN`.

- If the provider is already in the plugin's built-in defaults or can be imported from the oEmbed Directory browser (the "Browse oEmbed Directory" button in the admin), suggest that simpler path instead of manual configuration.
- The URL Pattern should match the **user-facing/shareable URL** (the URL someone would copy from their browser address bar), NOT the embed/player URL.
- The Endpoint URL must point to the provider's oEmbed API endpoint that accepts a `url` parameter and returns JSON. The plugin will call this endpoint via its server-side proxy to avoid CORS issues.
- Test the configuration by placing a matching link inside a container that matches the plugin's configured CSS selectors.

When oEmbed is not available:

If the provider does not offer an oEmbed endpoint, inform the user clearly and suggest:

1. Check if a third-party oEmbed proxy service (like Iframely or Embedly) supports the provider.
2. The provider may support oEmbed discovery — look for `<link type="application/json+oembed">` in the HTML head of content pages.
3. If none of these work, this plugin cannot embed content from that provider, as it relies on the oEmbed protocol.