WordPress

All about the mighty WP...

- SwupJS
  - SWUP Documentation
  - Comprehensive Guide to SwupJS Integration for WordPress/Bricks Builder
  - Umfassender Leitfaden zur SwupJS-Integration für WordPress/Bricks Builder
- Typografie in WP
  - wptexturize() in WordPress
  - Plugin zum Steuern von wptexturize()
  - wptexturize() in BricksBuilder & Polylang
- Security
  - wp-content und wp-includes zu schützen?
- Performance
  - WordPress-Plugins für Server-Auslastung in der Adminleiste

SwupJS

Transition animations, preloading + Bricks

SWUP Documentation

Introduction

Swup is a versatile page transition library that transforms multi-page websites into smooth, app-like experiences. Instead of letting the browser perform full page loads, swup intercepts link clicks, fetches new pages in the background, and animates the transition between old and new content.

Key Features

- **Smooth Transitions**: Animate between pages using CSS, JavaScript, or native View Transitions
- **Intelligent Caching**: Previously visited pages load instantly from memory
- **Browser History Support**: Back/forward navigation works as expected with scroll position restoration
- **Modular Architecture**: Extend functionality through a rich plugin ecosystem
- **Lightweight Core**: Small footprint with optional features added via plugins

---

Installation

Using a Bundler (Recommended)

```
npm install swup
```

```
import Swup from 'swup';
const swup = new Swup();
```

Using a CDN

```
<script src="https://unpkg.com/swup@4"></script>
<script>
  const swup = new Swup();
</script>
```

ES Modules with Fallback

For modern browsers with a fallback for older ones:

```
<!-- ES Module for modern browsers -->
<script type="module">
  import Swup from 'https://unpkg.com/swup@4?module';
  const swup = new Swup();
</script>

<!-- UMD fallback for older browsers -->
<script nomodule src="https://unpkg.com/swup@4"></script>
<script nomodule>
  const swup = new Swup();
</script>
```

Quick Start

Swup requires two things: a content container and transition styles.

1. Mark Your Content Container

Add the `#swup` id and a `transition-*` class to your main content element:

```
<main id="swup" class="transition-fade">
  <h1>Welcome</h1>
  <p>Lorem ipsum dolor sit amet.</p>
</main>
```

2. Define Transition Styles

```
/* Base transition styles */
html.is-changing .transition-fade {
  transition: opacity 0.25s;
  opacity: 1;
}

/* Styles for the leaving/entering state */
html.is-animating .transition-fade {
  opacity: 0;
}
```

3. Initialize Swup

```
import Swup from 'swup';
const swup = new Swup();
```

Complete Example

```
<!DOCTYPE html>
<html>
<head>
  <title>Swup Example</title>
  <style>
    html.is-changing .transition-fade {
      transition: opacity 0.25s;
      opacity: 1;
    }
    html.is-animating .transition-fade {
      opacity: 0;
    }
  </style>
</head>
<body>
  <main id="swup" class="transition-fade">
    <h1>Welcome</h1>
    <p>Lorem ipsum dolor sit amet.</p>
  </main>
```

```
  <script type="module">
    import Swup from 'https://unpkg.com/swup@4?module';
    const swup = new Swup();
  </script>
</body>
</html>
```

Core Concepts

Content Containers

Swup replaces only designated content containers, not the entire page. By default, it looks for an element with `id="swup"`. You can configure additional containers for elements like navigation or sidebars that need updating without animation.

```
const swup = new Swup({
  containers: ['#swup', '#sidebar', '#nav']
});
```

> **Note**: Only elements inside `<body>` are supported. Each selector should match a unique element.

Animation Classes

Swup applies classes to the `<html>` element during transitions:

Class	Description
`is-changing`	Present throughout the entire transition
`is-animating`	Present while content is animating (removed after content swap)
`is-leaving`	Present during the leave animation phase
`is-rendering`	Present during the enter animation phase

Class	Description
`to-[name]`	Added when using custom animations via `data-swup-animation`

Browser History

Swup integrates with the browser's History API. The URL always reflects the current page, and forward/backward navigation works naturally with scroll position restoration.

Scroll Behavior

Swup mimics native browser scrolling:

- Scroll position resets to top between pages
- Anchor links scroll to the target element
- History navigation restores previous scroll positions

Configuration Options

Pass options when initializing swup:

```
const swup = new Swup({
  // options here
});
```

Complete Options Reference

Option	Type	Default	Description
`animationSelector`	`string`	`'[class*="transition-"]'`	Selector for elements swup waits to finish animating
`animationScope`	`string`	`'html'`	Where animation classes are applied (`'html'` or `'containers'`)
`containers`	`array`	`['#swup']`	Content containers to replace

Option	Type	Default	Description
`cache`	`boolean`	`true`	Enable/disable page caching
`hooks`	`object`	`{}`	Register hook handlers at initialization
`ignoreVisit`	`function`	See below	Callback to ignore specific visits
`linkSelector`	`string`	`'a[href]'`	Selector for links that trigger transitions
`linkToSelf`	`string`	`'scroll'`	Behavior for links to current page (`'scroll'` or `'navigate'`)
`native`	`boolean`	`false`	Enable native View Transitions API
`plugins`	`array`	`[]`	Plugins to enable
`resolveUrl`	`function`	`(url) => url`	Transform URLs before loading
`requestHeaders`	`object`	See below	Custom headers for fetch requests
`skipPopStateHandling`	`function`	See below	Control history navigation handling
`timeout`	`number`	`0`	Fetch timeout in milliseconds (0 = disabled)
`animateHistoryBrowsing`	`boolean`	`false`	Animate back/forward navigation

Default Values

```
const swup = new Swup({
  animateHistoryBrowsing: false,
  animationSelector: '[class*="transition-"]',
  animationScope: 'html',
  cache: true,
  containers: ['#swup'],
  hooks: {},
  ignoreVisit: (url, { el } = {}) => el?.closest('[data-no-swup]'),
  linkSelector: 'a[href]',
  linkToSelf: 'scroll',
  native: false,
  plugins: [],
```

```
  resolveUrl: (url) => url,
  requestHeaders: {
    'X-Requested-With': 'swup',
    'Accept': 'text/html, application/xhtml+xml'
  },
  skipPopStateHandling: (event) => event.state?.source !== 'swup',
  timeout: 0
});
```

Animations

Swup supports three animation methods: CSS transitions, JavaScript animations, and native View Transitions.

CSS Animations

The default approach. Swup waits for CSS transitions and animations on elements with `transition-*` classes.

```
html.is-changing .transition-fade {
  transition: opacity 0.25s;
  opacity: 1;
}

html.is-animating .transition-fade {
  opacity: 0;
}
```

Different Leave and Enter Animations

Use `is-leaving` and `is-rendering` classes for asymmetric animations:

```
html.is-leaving .transition-slide {
  transform: translateX(-100%);
}

html.is-rendering .transition-slide {
```

```
  transform: translateX(100%);
}
```

JavaScript Animations

Use hooks to manage animations with libraries like GSAP:

```
import gsap from 'gsap';

swup.hooks.replace('animation:out:await', async () => {
  await gsap.to('.transition-fade', { opacity: 0, duration: 0.25 });
});

swup.hooks.replace('animation:in:await', async () => {
  await gsap.fromTo('.transition-fade',
    { opacity: 0 },
    { opacity: 1, duration: 0.25 }
  );
});
```

Native View Transitions

Enable the browser's View Transitions API for better performance:

```
const swup = new Swup({ native: true });
```

```
html.is-changing .transition-fade {
  view-transition-name: main;
}

::view-transition-old(main) {
  animation: fade 0.5s ease-in-out both;
}

::view-transition-new(main) {
  animation: fade 0.5s ease-in-out both reverse;
}
```

```
@keyframes fade {
  from { opacity: 1; }
  to { opacity: 0; }
}
```

Fallback for Unsupported Browsers

```
html.is-changing:not(.swup-native) .transition-fade {
  transition: 0.25s opacity;
  opacity: 1;
}

html.is-animating:not(.swup-native) .transition-fade {
  opacity: 0;
}
```

Disabling Animations

```
const swup = new Swup({ animationSelector: false });
```

Lifecycle & Hooks

Hooks allow you to execute custom code at specific points during page transitions.

Lifecycle Diagram

```
Link clicked → visit:start → page:load → content:replace → visit:end
                    ↓                         ↓
           animation:out:start          animation:in:start
                    ↓                         ↓
           animation:out:await          animation:in:await
                    ↓                         ↓
           animation:out:end            animation:in:end
```

Registering Handlers

```
swup.hooks.on('page:view', (visit) => {
  console.log('New page loaded:', visit.to.url);
});
```

Handler Options

```
// Execute once then remove
swup.hooks.on('page:view', handler, { once: true });
// or
swup.hooks.once('page:view', handler);

// Execute before internal handler
swup.hooks.on('content:replace', handler, { before: true });
// or
swup.hooks.before('content:replace', handler);

// Control execution order (negative = earlier, positive = later)
swup.hooks.on('visit:start', handler, { priority: -100 });
```

Async Handlers

Handlers can return Promises to pause execution:

```
swup.hooks.on('visit:start', async () => {
  await someAsyncOperation();
});
```

Removing Handlers

```
const handler = () => console.log('Page loaded');
swup.hooks.on('page:view', handler);
swup.hooks.off('page:view', handler);
```

Available Hooks

Hook	Description
`visit:start`	Transition begins
`visit:end`	Transition complete
`visit:abort`	Visit cancelled (new link clicked)
`page:load`	Page loaded (from cache or network)
`page:view`	New content is visible
`content:replace`	Old content replaced with new
`content:scroll`	Scroll position reset
`animation:out:start`	Leave animation begins
`animation:out:await`	Waiting for leave animation
`animation:out:end`	Leave animation complete
`animation:in:start`	Enter animation begins
`animation:in:await`	Waiting for enter animation
`animation:in:end`	Enter animation complete
`animation:skip`	Animations skipped (history navigation)
`link:click`	Link clicked
`link:self`	Link to current page clicked
`link:anchor`	Anchor link clicked
`link:newtab`	Link opens new tab
`history:popstate`	Back/forward button pressed
`fetch:request`	Fetch request sent
`fetch:error`	Fetch failed
`fetch:timeout`	Fetch timed out
`cache:set`	Page cached
`cache:clear`	Cache cleared
`scroll:top`	Scrolling to top
`scroll:anchor`	Scrolling to anchor
`enable`	Swup initialized
`disable`	Swup destroyed

Registering Hooks at Initialization

```
const swup = new Swup({
  hooks: {
    'visit:start': () => console.log('Starting'),
    'page:view': () => console.log('Loaded'),
    'content:replace.before': () => console.log('Before replace'),
    'visit:end.once': () => console.log('First visit only')
  }
});
```

DOM Events

Hooks are also dispatched as DOM events:

```
document.addEventListener('swup:page:view', ({ detail: { visit } }) => {
  console.log('Going to', visit.to.url);
});
```

The Visit Object

The visit object contains information about the current navigation and is available in all hook handlers.

Accessing the Visit Object

```
swup.hooks.on('page:view', (visit) => {
  console.log('Navigating to:', visit.to.url);
});
```

Structure

```
{
  id: 1042739,                    // Unique visit identifier
  from: {
    url: '/home',
```

```
    hash: ''
  },
  to: {
    url: '/about',
    hash: '#anchor',
    html: undefined,              // Populated after fetch
    document: undefined           // Populated after fetch
  },
  containers: ['#swup'],
  animation: {
    animate: true,
    name: 'fade'
  },
  trigger: {
    el: HTMLAnchorElement,        // undefined if via API
    event: MouseEvent             // undefined if via API
  },
  cache: {
    read: true,
    write: true
  },
  history: {
    action: 'push',
    popstate: false,
    direction: undefined          // 'forwards' | 'backwards' | undefined
  },
  scroll: {
    reset: true,
    target: '#anchor'
  },
  meta: {}                        // Custom metadata
}
```

Modifying Visit Behavior

Best done in the `visit:start` hook before requests or animations begin:

```
swup.hooks.on('visit:start', (visit) => {
  // Disable animations
```

```
  visit.animation.animate = false;

  // Set custom animation
  visit.animation.name = 'slide';

  // Keep scroll position
  visit.scroll.reset = false;

  // Change containers for this visit
  visit.containers = ['#sidebar'];

  // Replace instead of push history
  visit.history.action = 'replace';

  // Disable cache
  visit.cache.read = false;
  visit.cache.write = false;

  // Add custom metadata
  visit.meta.customData = 'value';
});
```

Detecting History Navigation

```
swup.hooks.on('visit:start', (visit) => {
  if (visit.history.popstate) {
    console.log('Direction:', visit.history.direction);
  }
});
```

Working with the Loaded Document

```
swup.hooks.on('content:replace', (visit) => {
  const lang = visit.to.document?.documentElement.getAttribute('lang');
  if (lang) {
    document.documentElement.setAttribute('lang', lang);
  }
});
```

Cache

Swup caches loaded pages in memory for instant repeat visits.

Disabling Cache

```
const swup = new Swup({ cache: false });
```

Cache API

```
// Check cache size
swup.cache.size; // 13

// Check if URL is cached
swup.cache.has('/about'); // true/false

// Get cached page
const page = swup.cache.get('/about');

// Add to cache
swup.cache.set('/about', {
  url: '/about',
  html: '<html>...</html>'
});

// Update cache entry
swup.cache.update('/about', { customData: 'value' });

// Delete specific entry
swup.cache.delete('/about');

// Clear all entries
swup.cache.clear();

// Remove entries matching condition
```

```
swup.cache.prune((url, page) => {
  return url.includes('/blog/');
});
```

Cache Pruning Strategies

Limit by count:

```
const maxEntries = 20;
swup.hooks.on('cache:set', () => {
  const urls = [...swup.cache.all.keys()].reverse().slice(maxEntries);
  swup.cache.prune((url) => urls.includes(url));
});
```

Limit by time (TTL):

```
const ttl = 5 * 60_000; // 5 minutes

swup.hooks.on('cache:set', (visit, { page }) => {
  swup.cache.update(page.url, { created: Date.now(), ttl });
});

swup.hooks.before('page:load', () => {
  swup.cache.prune((url, { created, ttl }) => Date.now() > created + ttl);
});
```

Markup Attributes

Control swup behavior with data attributes.

Ignoring Links

```
<!-- Ignore single link -->
<a href="/" data-no-swup>Ignored</a>

<!-- Ignore all links in container -->
```

```
<nav data-no-swup>
  <a href="/page1">Ignored</a>
  <a href="/page2">Ignored</a>
</nav>
```

Custom Animations

```
<a href="/about" data-swup-animation="slide">Slide Animation</a>
```

Adds `to-slide` class to HTML during transition:

```
html.is-changing.to-slide .transition-page {
  /* slide-specific styles */
}
```

Apply to all links in a container:

```
<nav data-swup-animation="slide">
  <a href="/page1">Slides</a>
  <a href="/page2">Slides</a>
  <a href="/page3" data-swup-animation="fade">Fades (override)</a>
</nav>
```

Persisting Elements

Keep element state (playback, scroll, event listeners) between pages:

```
<video src="/video.mp4" autoplay data-swup-persist="video-player"></video>
```

Elements are matched by their persist attribute value.

History Mode

Replace instead of push history entry:

```
<a href="/wizard-step-2" data-swup-history="replace">Next Step</a>
```

Methods & Helpers

Instance Methods

Navigate programmatically:

```
swup.navigate('/about');

// With options
swup.navigate('/about', {
  animate: false,
  animation: 'slide',
  history: 'replace',
  method: 'POST',
  data: new FormData(),
  cache: { read: false, write: true },
  meta: { customData: 'value' }
});
```

Destroy swup:

```
swup.destroy();
```

Manage plugins:

```
swup.use(new SwupScrollPlugin());
swup.unuse('SwupScrollPlugin');
const plugin = swup.findPlugin('SwupScrollPlugin');
```

Debug logging:

```
swup.log('Message', { data: 'object' });
```

Instance Properties

```
swup.options;   // Current configuration
swup.plugins;   // Active plugin instances
```

```
swup.location;  // Current location object

// Location details
swup.location.href;     // https://example.com/path?query#hash
swup.location.url;      // /path?query
swup.location.pathname; // /path
swup.location.search;   // ?query
swup.location.hash;     // #hash
```

Helper Utilities

```
import {
  Location,
  classify,
  createHistoryRecord,
  updateHistoryRecord,
  delegateEvent,
  getCurrentUrl
} from 'swup';

// Parse URLs
const { url, hash } = Location.fromUrl('http://example.com/about#section');
const { url, hash } = Location.fromElement(linkElement);

// Sanitize strings for CSS/slugs
classify('Hello World'); // 'hello-world'

// Manage history
createHistoryRecord('/new-page', { custom: 'data' });
updateHistoryRecord(null, { custom: 'data' });

// Delegate events
const delegation = delegateEvent('form', 'submit', (event) => {
  console.log('Form submitted');
});
delegation.destroy(); // Clean up
```

Plugins

Swup's modular architecture allows extending functionality through plugins.

Installing Plugins

```
npm install @swup/scroll-plugin
```

```
import Swup from 'swup';
import SwupScrollPlugin from '@swup/scroll-plugin';

const swup = new Swup({
  plugins: [new SwupScrollPlugin()]
});
```

Or add/remove plugins dynamically:

```
swup.use(new SwupScrollPlugin());
swup.unuse('SwupScrollPlugin');
```

Official Plugins

Plugin	Description
Accessibility Plugin	Screen reader announcements, focus management, reduced motion support
Body Class Plugin	Update body classname on navigation
Debug Plugin	Development logging and warnings
Forms Plugin	Handle form submissions with transitions
Fragment Plugin	Dynamic container replacement rules
Head Plugin	Update `<head>` contents
JS Plugin	Advanced JavaScript animation management
Morph Plugin	Morph elements instead of replacing
Parallel Plugin	Animate old and new content simultaneously
Preload Plugin	Preload pages on hover/visibility
Progress Bar Plugin	Loading progress indicator

Plugin	Description
Route Name Plugin	Named routes and route-based animations
Scripts Plugin	Re-run scripts after navigation
Scroll Plugin	Smooth scrolling with offsets

Official Themes

Theme	Description
Fade Theme	Fade transitions
Slide Theme	Slide transitions
Overlay Theme	Overlay slide transitions

Plugin Examples

Accessibility Plugin

```
import SwupA11yPlugin from '@swup/a11y-plugin';

const swup = new Swup({
  plugins: [new SwupA11yPlugin({
    headingSelector: ['main h1', 'h1'],
    respectReducedMotion: true,
    autofocus: false,
    announcements: {
      visit: 'Navigated to: {title}',
      url: 'New page at {url}'
    }
  })]
});
```

Preload Plugin

```
import SwupPreloadPlugin from '@swup/preload-plugin';

const swup = new Swup({
  plugins: [new SwupPreloadPlugin({
```

```
    throttle: 5,
    preloadHoveredLinks: true,
    preloadVisibleLinks: {
      threshold: 0.2,
      delay: 500
    }
  })]
});
```

Mark specific links for preloading:

```
<a href="/about" data-swup-preload>Preloaded on page load</a>

<nav data-swup-preload-all>
  <a href="/page1">All preloaded</a>
  <a href="/page2">All preloaded</a>
</nav>
```

Troubleshooting

Scripts Not Running

Swup doesn't execute scripts in new content by default. Solutions:

1. Use hooks to trigger code after navigation
2. Use the **Head Plugin** for scripts in `<head>`
3. Use the **Scripts Plugin** to re-run scripts

Stylesheets Not Loading

Swup doesn't update `<head>` by default. Either:

- Use a single stylesheet for the entire site
- Install the **Head Plugin** with `awaitAssets: true`

Screen Readers Not Announcing Changes

Install the **Accessibility Plugin** for proper announcements.

Scroll Anchors Hidden by Fixed Header

```
[id] {
  scroll-margin-top: 100px; /* Adjust for header height */
}
```

Or use the **Scroll Plugin** with offset option.

History Navigation Not Working

Swup only handles history entries it created. Check:

- Your `linkSelector` includes all page links
- You're using swup's history helpers for custom entries
- Configure `skipPopStateHandling` to accept all entries if needed

Hard Page Load Instead of Transition

Common causes:

1. Missing container on old or new page
2. Visit ignored by `ignoreVisit` callback or `data-no-swup`
3. Click event propagation stopped before reaching swup

Class Name Conflicts

If another library uses `transition-*` classes:

```
const swup = new Swup({
  animationSelector: '[class*="swup-transition-"]'
});
```

```
<main id="swup" class="swup-transition-fade"></main>
```

Browser Support

Full Support (CDN)

Chrome	Edge	Safari	Safari iOS	Firefox
80+	80+	13.1+	13.4+	74+

Extended Support

- **Transpiling** with Babel: Chrome 66+, Edge 16+, Safari 13+, Firefox 60+
- **Polyfills** via polyfill.io: Further extends support
- **Swup v3**: For IE 11 support (at cost of newer features)

Upgrading from v3

Breaking Changes Summary

1. **Install v4**: `npm install swup@latest`
2. **Hooks replace events**: `swup.on()` → `swup.hooks.on()`
3. **Hook names changed**: `pageView` → `page:view`
4. **Visit object replaces transition**: `swup.transition` → `visit` parameter
5. **Navigate method renamed**: `swup.loadPage()` → `swup.navigate()`
6. **Animation attribute renamed**: `data-swup-transition` → `data-swup-animation`
7. **Unique container selectors required**
8. **Scroll support built-in** (Scroll Plugin optional)

Migration Examples

Events to Hooks:

```
// v3
swup.on('pageView', () => {});
swup.off('pageView', handler);
```

```
// v4
swup.hooks.on('page:view', () => {});
swup.hooks.off('page:view', handler);
```

Before/After Hooks:

```
// v3
swup.on('willReplaceContent', () => {});
swup.on('contentReplaced', () => {});

// v4
swup.hooks.before('content:replace', () => {});
swup.hooks.on('content:replace', () => {});
```

Transition Object:

```
// v3
swup.on('transitionStart', () => {
  console.log(swup.transition.to);
  console.log(swup.transition.custom);
});

// v4
swup.hooks.on('visit:start', (visit) => {
  console.log(visit.to.url);
  console.log(visit.animation.name);
});
```

Cache API:

```
// v3
swup.cache.cacheUrl({
  url: '/about',
  title: 'About',
  blocks: ['<div id="swup"></div>'],
  originalContent: '<html>...</html>'
});

// v4
swup.cache.set('/about', {
  url: '/about',
```

```
  html: '<html>...</html>'
});
```

Navigation:

```
// v3
swup.loadPage({ url: '/about' });

// v4
swup.navigate('/about');
```

Animation Attribute:

```
<!-- v3 -->
<a href="/about" data-swup-transition="slide">About</a>

<!-- v4 -->
<a href="/about" data-swup-animation="slide">About</a>
```

Comprehensive Guide to SwupJS Integration for WordPress/Bricks Builder

Technical Background

What is Swup?

Swup is a JavaScript library that transforms traditional multi-page websites into Single Page Application (SPA)-like experiences. Instead of performing full browser page reloads when clicking links, Swup:

1. **Intercepts link clicks** - Captures navigation events before the browser handles them
2. **Fetches pages via AJAX** - Loads new page content in the background using the Fetch API
3. **Replaces content dynamically** - Swaps only the designated content containers, preserving the rest of the DOM
4. **Animates transitions** - Applies CSS classes during the transition for smooth visual effects
5. **Manages browser history** - Uses the History API to maintain proper URL states and back/forward navigation

Why Use Swup with Bricks Builder?

Bricks Builder generates standard WordPress pages with full HTML documents. Without Swup, each page navigation:

- Triggers a complete browser reload
- Re-parses all HTML, CSS, and JavaScript
- Reinitializes all scripts and components
- Creates a visible "flash" between pages

With Swup, your Bricks-built site gains:

- **Perceived performance improvement** - Pages appear to load instantly
- **Smooth visual transitions** - Professional fade, slide, or scale effects
- **Reduced server load** - Cached pages load from memory
- **Better user experience** - App-like navigation feel

How the Script Works

The swup-bricks-integration.js file is a self-contained integration that:

1. **Loads Swup and plugins from CDN** - No npm installation required
2. **Injects transition CSS automatically** - Styles are added to the document head
3. **Configures all options in one place** - The SWUP_CONFIG object centralizes settings
4. **Handles WordPress/Bricks specifics** - Excludes admin links, handles common plugins
5. **Provides reinitalization hooks** - The reinitializeScripts function handles third-party scripts

---

Installation Instructions

Method 1: Bricks Settings (Recommended)

1. **Access Bricks Settings**
   - Navigate to WordPress Admin → Bricks → Settings
   - Go to the "Custom Code" tab
2. **Add the Script**
   - Locate "Body (footer) scripts" section
   - Paste the entire contents of swup-bricks-integration.js wrapped in `<script>` tags:

```
<script>
// Paste the entire script content here
</script>
```

3. **Save Settings**
   - Click "Save Settings" at the bottom of the page

Method 2: Code Element in Template

1. **Edit Your Header/Footer Template**
   - Open Bricks Builder on your header or footer template
   - Add a "Code" element at the end of the template
2. **Configure the Code Element**
   - Set the rendering to "Execute Code"
   - Paste the script with `<script>` tags
3. **Save and Publish**
   - Save the template and publish changes

Method 3: Child Theme (Advanced)

1. **Create or Edit functions.php**

```
function enqueue_swup_integration() {
    wp_enqueue_script(
        'swup-bricks',
        get_stylesheet_directory_uri() . '/js/swup-bricks-integration.js',
        array(),
        '1.0.0',
        true // Load in footer
    );
}
add_action('wp_enqueue_scripts', 'enqueue_swup_integration');
```

2. **Place the Script File**
   - Create a `js` folder in your child theme
   - Save the script as `swup-bricks-integration.js`

Configuration Guide

Essential Settings

Content Container

The most critical setting is containers. This defines which parts of the page Swup will replace:

```
containers: ['#brx-content'],
```

Technical explanation: Swup finds elements matching these selectors on both the current page and the fetched page, then replaces the current element's innerHTML with the new content. The default `#brx-content` is Bricks' main content wrapper.

Important requirements:

- The container must exist on **every page** of your site
- The selector must match **exactly one element** per page
- The container should be **inside** `<body>`

Common Bricks containers:

- `#brx-content` - Main content area (default)
- `#brx-header` - If your header changes between pages
- `.custom-sidebar` - If you have a dynamic sidebar

Animation Selector

```
animationSelector: '[class*="transition-"]',
```

Technical explanation: This CSS attribute selector matches any element with a class containing "transition-". Swup monitors these elements for CSS `transitionend` and `animationend` events to know when animations complete.

How to use:

1. Add `transition-fade`, `transition-slide`, or `transition-scale` class to your content wrapper in Bricks
2. The injected CSS handles the actual animations
3. Swup waits for these elements to finish animating before showing new content

Animation Settings

Native View Transitions

```
nativeViewTransitions: false,
```

Technical explanation: The View Transitions API is a new browser feature (Chrome 111+, Edge 111+) that provides hardware-accelerated transitions between page states. When enabled:

- Browser creates a snapshot of the current page
- New content is rendered off-screen
- Browser animates between snapshots using GPU acceleration

When to enable: Only if your target audience primarily uses Chrome/Edge and you want smoother animations.

Animation Scope

```
animationScope: 'html',
```

Options:

- `'html'` - Animation classes (`is-changing`, `is-animating`, etc.) are applied to `<html>` element
- `'containers'` - Classes are applied directly to container elements

Recommendation: Keep as `'html'` for easier CSS targeting with selectors like `html.is-animating .transition-fade`.

Cache Configuration

Cache TTL (Time-To-Live)

```
cacheTTL: 5 * 60 * 1000, // 5 minutes in milliseconds
```

Technical explanation: Swup stores fetched page HTML in a JavaScript Map object in memory. The TTL determines how long cached pages remain valid. After expiration, Swup fetches a fresh copy.

Considerations:

- **Dynamic content sites**: Lower TTL (1-5 minutes) ensures content freshness
- **Static content sites**: Higher TTL (30-60 minutes) improves performance
- **Real-time content**: Set to `0` to disable caching entirely

Maximum Cache Entries

```
maxCacheEntries: 20,
```

Technical explanation: Limits memory usage by removing oldest cached pages when the limit is exceeded. This is implemented via the cache:set hook that prunes excess entries.

Preload Settings

```
preloadEnabled: true,
preloadOnHover: true,
preloadVisibleLinks: false,
preloadThrottle: 3,
```

How preloading works:

1. **Hover preload**: When user hovers over a link, Swup immediately fetches that page
2. **Visible links**: Intersection Observer API detects links entering the viewport
3. **Throttle**: Limits concurrent fetch requests to prevent server overload

Technical benefit: By the time the user clicks, the page is already cached, resulting in instant navigation.

WordPress-Specific Settings

Ignore Selectors

```
ignoreSelectors: [
  '[data-no-swup]',
  '#wpadminbar a',
  'a[href*="wp-admin"]',
  'a[href*="wp-login"]',
  'a[href*="/cart"]',
  // ... more
],
```

Technical explanation: The buildIgnoreVisit function creates a callback that checks every link click against these selectors. Matching links trigger normal browser navigation instead of Swup transitions.

Why ignore these:

- **Admin bar**: WordPress admin links should always perform full page loads
- **WooCommerce cart/checkout**: These pages often require full reloads for session management
- **Login pages**: Authentication typically requires traditional form submission

---

Adding Transitions to Your Bricks Site

Step 1: Add Transition Class to Content

1. **Open Bricks Builder** on your main template (usually "Single" or "Archive")
2. **Select your main content wrapper** (the element containing page content)
3. **Go to Style → CSS Classes**
4. **Add class**: `transition-fade`

Alternatively, target the default Bricks content wrapper in your CSS:

```
#brx-content {
  /* This becomes the animated element */
}
```

And modify the script's animationSelector:

```
animationSelector: '#brx-content',
```

Step 2: Customize Transition Styles

The script injects default styles via the TRANSITION_CSS constant. You can override these in Bricks' custom CSS:

```
/* Slower fade transition */
html.is-changing .transition-fade {
  transition: opacity 0.5s ease-in-out;
}

/* Custom slide direction */
html.is-leaving .transition-slide {
  transform: translateX(-100%);
}

html.is-rendering .transition-slide {
```

```
  transform: translateX(100%);
}
```

Step 3: Test the Transitions

1. **View your site** (not in Bricks editor)
2. **Open browser DevTools** → Console
3. **Enable debug mode** in the script:

   ```
   debug: true,
   ```

4. **Click links** and observe console logs showing transition lifecycle

---

Handling Third-Party Scripts

The Reinitialization Problem

Technical background: When Swup replaces content, it injects new HTML but doesn't re-execute JavaScript. Any scripts that:

- Query the DOM for elements
- Attach event listeners
- Initialize components (sliders, lightboxes, etc.)

...will not work on the new content because they only ran once on initial page load.

Solution: The reinitializeScripts Function

The reinitializeScripts function is called after every page transition:

```
function reinitializeScripts() {
  // Bricks frontend JS
  if (typeof bricksInit === 'function') {
    bricksInit();
  }

  // Custom event for your scripts
```

```
  document.dispatchEvent(new CustomEvent('swup:contentReplaced', {
    detail: {swup: window.swup}
  }));

  // Third-party reinitializations...
}
```

Adding Your Own Reinitializations

Method 1: Modify the Function

Add your code directly to reinitializeScripts:

```
function reinitializeScripts() {
  // ... existing code ...

  // Your custom initialization
  if (typeof MyCustomSlider !== 'undefined') {
    MyCustomSlider.init();
  }
}
```

Method 2: Listen for Custom Event

In a separate script:

```
document.addEventListener('swup:contentReplaced', function(event) {
  // Reinitialize your scripts here
  initializeMyComponents();
});
```

Method 3: Use the onPageView Callback

In the SWUP_CONFIG object:

```
onPageView: function(visit) {
  // Called after every page transition
  console.log('Navigated to:', visit.to.url);
```

```
  // Reinitialize components
  initializeLightboxes();
  initializeSliders();
}
```

Analytics Integration

Google Analytics 4

The script includes GA4 tracking in onPageView:

```
if (typeof gtag !== 'undefined') {
  gtag('config', 'GA_MEASUREMENT_ID', {
    page_path: visit.to.url
  });
}
```

Important: Replace `GA_MEASUREMENT_ID` with your actual measurement ID (e.g., `G-XXXXXXXXXX`).

Matomo/Piwik

Also included:

```
if (typeof _paq !== 'undefined') {
  _paq.push(['setCustomUrl', visit.to.url]);
  _paq.push(['trackPageView']);
}
```

Other Analytics Platforms

Add similar code to onPageView:

```
// Facebook Pixel
if (typeof fbq !== 'undefined') {
  fbq('track', 'PageView');
```

```
}

// Plausible
if (typeof plausible !== 'undefined') {
  plausible('pageview');
}
```

Troubleshooting

Issue: Hard page reload instead of transition

Causes:

1. **Container not found**: Ensure `#brx-content` exists on all pages
2. **Link is ignored**: Check if the link matches any ignoreSelectors
3. **External link**: Swup only handles same-origin links

Debug steps:

1. Enable `debug: true`
2. Check console for Swup logs
3. Verify container exists: `document.querySelector('#brx-content')`

Issue: Animations not working

Causes:

1. **Missing transition class**: Add `transition-fade` to content wrapper
2. **CSS not injected**: Check if `#swup-transitions` style element exists in `<head>`
3. **Animation selector mismatch**: Ensure your class matches `animationSelector` pattern

Debug steps:

1. Inspect `<html>` element during navigation for `is-changing`, `is-animating` classes
2. Check computed styles on transition elements

Issue: Scripts not running after navigation

Cause: JavaScript only executes on initial page load.

Solution: Add reinitialization code to reinitializeScripts or listen for `swup:contentReplaced` event.

Issue: Forms not submitting properly

Cause: Swup doesn't handle form submissions by default.

Solutions:

1. Add `data-no-swup` to the form's submit link/button
2. Install the Swup Forms Plugin (add to CDN URLs and enable)

Issue: Scroll position issues with fixed header

Solution: Adjust scrollOffset:

```
scrollOffset: 80, // Height of your fixed header in pixels
```

Performance Optimization

Recommended Production Settings

```
const SWUP_CONFIG = {
  // Disable debug logging
  debug: false,

  // Optimize caching
  cacheEnabled: true,
  maxCacheEntries: 30,
  cacheTTL: 10 * 60 * 1000, // 10 minutes
```

```
  // Enable preloading
  preloadEnabled: true,
  preloadOnHover: true,
  preloadThrottle: 5,

  // Disable unused plugins
  scriptsEnabled: false, // Only enable if absolutely needed
  progressBarEnabled: true,

  // Enable native transitions for supported browsers
  nativeViewTransitions: true,
};
```

Measuring Impact

Use browser DevTools to compare:

1. **Network waterfall**: Compare full page load vs. Swup navigation
2. **Performance timeline**: Measure time-to-interactive
3. **Core Web Vitals**: Monitor LCP, FID, CLS before/after implementation

---

Advanced Customization

Custom Animation per Link

Use `data-swup-animation` attribute:

```
<a href="/gallery" data-swup-animation="slide">View Gallery</a>
```

This adds `to-slide` class to `<html>` during transition. Create corresponding CSS:

```
html.is-changing.to-slide .transition-fade {
  transition: transform 0.4s ease;
}
```

```
html.is-leaving.to-slide .transition-fade {
  transform: translateX(-100%);
}

html.is-rendering.to-slide .transition-fade {
  transform: translateX(100%);
}
```

Programmatic Navigation

Access the Swup instance globally:

```
// Navigate to a page
window.swup.navigate('/about');

// Navigate without animation
window.swup.navigate('/contact', { animate: false });

// Clear cache
window.swup.cache.clear();
```

Excluding Specific Pages

Dynamically ignore pages based on URL:

```
ignoreVisit: function(url, { el } = {}) {
  // Ignore checkout pages
  if (url.includes('/checkout/')) return true;

  // Ignore specific page
  if (url === '/special-page/') return true;

  // Check element against ignore selectors (default behavior)
  const { ignoreSelectors } = SWUP_CONFIG;
  if (el) {
    for (const selector of ignoreSelectors) {
      if (el.closest(selector)) return true;
    }
```

```
    }

    return false;
}
```

This guide covers the complete setup and customization of the SwupJS integration for Bricks Builder. For further questions or issues, refer to the official Swup documentation or examine the detailed comments within the swup-bricks-integration.js file.

Umfassender Leitfaden zur SwupJS-Integration für WordPress/Bricks Builder

Technischer Hintergrund

Was ist Swup?

Swup ist eine JavaScript-Bibliothek, die traditionelle Multi-Page-Websites in Erlebnisse verwandelt, die sich wie Single Page Applications (SPAs) anfühlen. Anstatt bei Klicks auf Links vollständige Browser-Seitenreloads durchzuführen, macht Swup Folgendes:

1. **Fängt Link-Klicks ab** – Erfasst Navigationsereignisse, bevor der Browser sie verarbeitet
2. **Lädt Seiten per AJAX** – Lädt neue Seiteninhalte im Hintergrund über die Fetch API
3. **Ersetzt Inhalte dynamisch** – Tauscht nur die festgelegten Inhalts-Container aus und behält den Rest des DOM bei
4. **Animiert Übergänge** – Wendet während des Übergangs CSS-Klassen für flüssige visuelle Effekte an
5. **Verwaltet den Browser-Verlauf** – Nutzt die History API, um korrekte URL-Zustände sowie Zurück-/Vorwärtsnavigation zu erhalten

Warum Swup mit Bricks Builder verwenden?

Bricks Builder erzeugt standardmäßige WordPress-Seiten mit vollständigen HTML-Dokumenten. Ohne Swup führt jede Seitennavigation zu Folgendem:

- Ein vollständiger Browser-Reload wird ausgelöst
- HTML, CSS und JavaScript werden komplett neu eingelesen
- Alle Skripte und Komponenten werden neu initialisiert

- Zwischen Seiten entsteht ein sichtbares „Flackern“

Mit Swup erhält Ihre mit Bricks erstellte Website:

- **Verbesserte wahrgenommene Performance** – Seiten wirken, als würden sie sofort laden
- **Sanfte visuelle Übergänge** – Professionelle Fade-, Slide- oder Scale-Effekte
- **Reduzierte Serverlast** – Zwischengespeicherte Seiten werden aus dem Speicher geladen
- **Bessere Benutzererfahrung** – App-ähnliches Navigationsgefühl

Wie das Skript funktioniert

Die Datei `swup-bricks-integration.js` ist eine eigenständige Integration, die:

1. **Swup und Plugins von einem CDN lädt** – Keine npm-Installation erforderlich
2. **Übergangs-CSS automatisch einfügt** – Styles werden dem Dokument-`head` hinzugefügt
3. **Alle Optionen zentral konfiguriert** – Das Objekt `SWUP_CONFIG` bündelt die Einstellungen
4. **WordPress-/Bricks-spezifische Besonderheiten behandelt** – Schließt Admin-Links aus, berücksichtigt häufige Plugins
5. **Hooks zur Reinitialisierung bereitstellt** – Die Funktion `reinitializeScripts` behandelt Skripte von Drittanbietern

---

Installationsanleitung

Methode 1: Bricks-Einstellungen (empfohlen)

1. Auf Bricks-Einstellungen zugreifen

Navigieren Sie zu **WordPress Admin → Bricks → Einstellungen**
Wechseln Sie zum Tab **„Custom Code“**

2. Das Skript hinzufügen

Suchen Sie den Bereich **„Body (footer) scripts“**
Fügen Sie den gesamten Inhalt von `swup-bricks-integration.js` ein, umschlossen von `<script>`-Tags:

3. Einstellungen speichern

Klicken Sie unten auf **„Save Settings“**

Methode 2: Code-Element im Template

1. Ihr Header-/Footer-Template bearbeiten

Öffnen Sie den Bricks Builder für Ihr Header- oder Footer-Template
Fügen Sie am Ende des Templates ein **„Code“**-Element hinzu

2. Das Code-Element konfigurieren

Setzen Sie das Rendering auf **„Execute Code“**
Fügen Sie das Skript mit `<script>`-Tags ein

3. Speichern und veröffentlichen

Speichern Sie das Template und veröffentlichen Sie die Änderungen

---

Methode 3: Child Theme (fortgeschritten)

1. `functions.php` erstellen oder bearbeiten

2. Die Skriptdatei einfügen

```
<script>
// Fügen Sie hier den gesamten Skriptinhalt ein
</script>
```

```
function enqueue_swup_integration() {
    wp_enqueue_script(
        'swup-bricks',
        get_stylesheet_directory_uri() . '/js/swup-bricks-integration.js',
        array(),
        '1.0.0',
        true // Im Footer laden
    );
```

```
}
add_action('wp_enqueue_scripts', 'enqueue_swup_integration');
```

Erstellen Sie einen Ordner `js` in Ihrem Child Theme
Speichern Sie das Skript als `swup-bricks-integration.js`

Konfigurationsleitfaden

Wesentliche Einstellungen

Inhalts-Container

Die wichtigste Einstellung ist `containers`. Sie definiert, welche Bereiche der Seite Swup ersetzen soll.

Technische Erklärung:
Swup sucht auf der aktuellen und auf der geladenen Seite nach Elementen, die diesen Selektoren entsprechen, und ersetzt dann das `innerHTML` des aktuellen Elements durch den neuen Inhalt. Der Standardwert `#brx-content` ist der Haupt-Content-Wrapper von Bricks.

Wichtige Anforderungen:

- Der Container muss auf **jeder Seite** Ihrer Website vorhanden sein
- Der Selektor darf pro Seite **genau ein Element** treffen
- Der Container sollte sich **innerhalb** von `<body>` befinden

Häufige Bricks-Container:

- `#brx-content` – Hauptinhaltsbereich (Standard)
- `#brx-header` – Falls sich Ihr Header zwischen Seiten ändert
- `.custom-sidebar` – Wenn Sie eine dynamische Sidebar haben

Animations-Selektor

Technische Erklärung:
Dieser CSS-Attributselektor trifft auf jedes Element zu, dessen Klasse `transition-` enthält. Swup überwacht diese Elemente auf `transitionend`- und `animationend`-Events, um zu erkennen, wann Animationen abgeschlossen sind.

Verwendung:

1. Fügen Sie `transition-fade`, `transition-slide` oder `transition-scale` Ihrem Content-Wrapper in Bricks hinzu
2. Das eingefügte CSS übernimmt die eigentlichen Animationen
3. Swup wartet, bis diese Elemente fertig animiert sind, bevor neuer Inhalt angezeigt wird

```
containers: ['#brx-content'],
animationSelector: '[class*="transition-"]',
```

Animationseinstellungen

Native View Transitions

Technische Erklärung:
Die View Transitions API ist eine neue Browser-Funktion (Chrome 111+, Edge 111+), die hardwarebeschleunigte Übergänge zwischen Seitenzuständen ermöglicht. Wenn aktiviert:

- Der Browser erstellt eine Momentaufnahme der aktuellen Seite
- Neuer Inhalt wird außerhalb des sichtbaren Bereichs gerendert
- Der Browser animiert GPU-beschleunigt zwischen den Zuständen

Wann aktivieren:
Nur wenn Ihre Zielgruppe überwiegend Chrome/Edge nutzt und Sie flüssigere Animationen möchten.

Animationsbereich

Optionen:

- `'html'` – Animationsklassen (`is-changing`, `is-animating` usw.) werden auf das `<html>`-Element angewendet
- `'containers'` – Klassen werden direkt auf die Container-Elemente angewendet

Empfehlung:
Bei `'html'` bleiben, da CSS-Selektoren wie `html.is-animating .transition-fade` einfacher zu verwenden sind.

Cache-Konfiguration

Cache-TTL (Time-To-Live)

Technische Erklärung:
Swup speichert geladenes Seiten-HTML in einem JavaScript-`Map`-Objekt im Speicher. Die TTL bestimmt, wie lange zwischengespeicherte Seiten gültig bleiben. Nach Ablauf lädt Swup eine frische Version.

Überlegungen:

- **Websites mit dynamischen Inhalten:** Niedrige TTL (1–5 Minuten) für aktuellere Inhalte
- **Websites mit statischen Inhalten:** Höhere TTL (30–60 Minuten) für bessere Performance
- **Echtzeit-Inhalte:** Auf `0` setzen, um Caching vollständig zu deaktivieren

Maximale Cache-Einträge

Technische Erklärung:
Begrenzt die Speichernutzung, indem die ältesten Cache-Einträge entfernt werden, sobald das Limit überschritten wird. Dies wird über den Hook `cache:set` umgesetzt.

```
nativeViewTransitions: false,
animationScope: 'html',
cacheTTL: 5 * 60 * 1000, // 5 Minuten in Millisekunden
maxCacheEntries: 20,
```

Preload-Einstellungen

Wie Preloading funktioniert:

1. **Hover-Preload:** Wenn der Benutzer über einen Link hovert, lädt Swup diese Seite sofort
2. **Sichtbare Links:** Die Intersection Observer API erkennt Links, die in den Viewport kommen
3. **Throttle:** Begrenzung gleichzeitiger Requests, um den Server nicht zu überlasten

Technischer Vorteil:
Bis der Benutzer klickt, liegt die Seite bereits im Cache und die Navigation wirkt sofort.

WordPress-spezifische Einstellungen

Ignore-Selektoren

Technische Erklärung:
Die Funktion `buildIgnoreVisit` erstellt einen Callback, der jeden Link-Klick gegen diese Selektoren prüft. Passende Links verwenden normale Browser-Navigation statt Swup-Übergänge.

Warum diese ignorieren:

- **Admin-Bar:** WordPress-Admin-Links sollten immer vollständige Seitenloads auslösen
- **WooCommerce Cart/Checkout:** Diese Seiten brauchen oft vollständige Reloads für Session-Management
- **Login-Seiten:** Authentifizierung erfordert typischerweise klassische Formularübermittlung

```
preloadEnabled: true,
preloadOnHover: true,
preloadVisibleLinks: false,
preloadThrottle: 3,

ignoreSelectors: [
  '[data-no-swup]',
  '#wpadminbar a',
  'a[href*="wp-admin"]',
  'a[href*="wp-login"]',
  'a[href*="/cart"]',
  // ... weitere
],
```

Übergänge zu Ihrer Bricks-Seite hinzufügen

Schritt 1: Übergangsklasse zum Inhalt hinzufügen

1. Öffnen Sie den **Bricks Builder** in Ihrem Haupttemplate (meist **„Single“** oder **„Archive“**)
2. Wählen Sie Ihren **Haupt-Content-Wrapper**
3. Gehen Sie zu **Style → CSS Classes**
4. Fügen Sie die Klasse `transition-fade` hinzu

Alternativ können Sie den Standard-Bricks-Content-Wrapper direkt in Ihrem CSS ansprechen:

```
#brx-content {
  /* Dies wird das animierte Element */
}
```

Und den `animationSelector` im Skript anpassen:

```
animationSelector: '#brx-content',
```

Schritt 2: Übergangsstile anpassen

Das Skript fügt Standard-Styles über die Konstante `TRANSITION_CSS` ein. Sie können diese in Bricks unter „Custom CSS“ überschreiben:

```
/* Langsamerer Fade-Übergang */
html .is-changing .transition-fade {
  transition: opacity 0.5s ease-in-out;
}

/* Benutzerdefinierte Slide-Richtung */
html .is-leaving .transition-slide {
  transform: translateX(-100%);
}
```

```
html .is-rendering .transition-slide {
  transform: translateX(100%);
}
```

Schritt 3: Übergänge testen

1. **Öffnen Sie Ihre Website** (nicht im Bricks-Editor)
2. Öffnen Sie die **Browser-DevTools → Konsole**
3. Aktivieren Sie den **Debug-Modus** im Skript:

```
debug: true,
```

4. Klicken Sie auf Links und beobachten Sie die Konsolenmeldungen zum Übergangsablauf

---

Umgang mit Drittanbieter-Skripten

Das Reinitialisierungsproblem

Technischer Hintergrund:
Wenn Swup Inhalte ersetzt, wird neues HTML eingefügt, aber JavaScript nicht erneut ausgeführt. Skripte, die:

- das DOM nach Elementen durchsuchen
- Event-Listener anhängen
- Komponenten initialisieren (Slider, Lightboxes usw.)

... funktionieren auf dem neuen Inhalt nicht mehr, da sie nur beim initialen Laden ausgeführt wurden.

Lösung: Die Funktion `reinitializeScripts`

Die Funktion `reinitializeScripts` wird nach jedem Seitenübergang aufgerufen:

```
function reinitializeScripts() {
  // Bricks Frontend-JS
  if (typeof bricksInit === 'function') {
```

```
    bricksInit();
  }

  // Benutzerdefiniertes Event für Ihre Skripte
  document.dispatchEvent(new CustomEvent('swup:contentReplaced', {
    detail: { swup: window.swup }
  }));

  // Reinitialisierungen von Drittanbietern...
}
```

Eigene Reinitialisierungen hinzufügen

Methode 1: Funktion direkt erweitern

```
function reinitializeScripts() {
  // ... bestehender Code ...

  // Ihre benutzerdefinierte Initialisierung
  if (typeof MyCustomSlider !== 'undefined') {
    MyCustomSlider.init();
  }
}
```

Methode 2: Auf ein benutzerdefiniertes Event hören

```
document.addEventListener('swup:contentReplaced', function(event) {
  // Ihre Skripte hier erneut initialisieren
  initializeMyComponents();
});
```

Methode 3: Den `onPageView`-Callback verwenden

Im Objekt `SWUP_CONFIG`:

```
onPageView: function(visit) {
  // Wird nach jedem Seitenübergang aufgerufen
  console.log('Navigiert zu:', visit.to.url);
```

```
  // Komponenten neu initialisieren
  initializeLightboxes();
  initializeSliders();
}
```

Analytics-Integration

Google Analytics 4

Das Skript enthält GA4-Tracking in `onPageView`:

```
if (typeof gtag !== 'undefined') {
  gtag('config', 'GA_MEASUREMENT_ID', {
    page_path: visit.to.url
  });
}
```

Wichtig:
Ersetzen Sie `GA_MEASUREMENT_ID` durch Ihre echte Measurement-ID (z. B. `G-XXXXXXXXXX`).

Matomo/Piwik

Ebenfalls enthalten:

```
if (typeof _paq !== 'undefined') {
  _paq.push(['setCustomUrl', visit.to.url]);
  _paq.push(['trackPageView']);
}
```

Weitere Analytics-Plattformen

Fügen Sie ähnlichen Code in `onPageView` ein:

```
// Facebook Pixel
if (typeof fbq !== 'undefined') {
  fbq('track', 'PageView');
```

```
}

// Plausible
if (typeof plausible !== 'undefined') {
  plausible('pageview');
}
```

Fehlerbehebung

Problem: Harter Seitenreload statt Übergang

Ursachen:

1. **Container nicht gefunden:** Stellen Sie sicher, dass `#brx-content` auf allen Seiten existiert
2. **Link wird ignoriert:** Prüfen Sie, ob der Link zu `ignoreSelectors` passt
3. **Externer Link:** Swup verarbeitet nur Links derselben Origin

Debug-Schritte:

1. `debug: true` aktivieren
2. Konsole auf Swup-Logs prüfen
3. Prüfen, ob der Container existiert: `document.querySelector('#brx-content')`

Problem: Animationen funktionieren nicht

Ursachen:

1. **Fehlende Übergangsklasse:** `transition-fade` zum Content-Wrapper hinzufügen
2. **CSS nicht eingefügt:** Prüfen, ob das Style-Element `#swup-transitions` im `<head>` existiert
3. **Animationsselektor stimmt nicht:** Sicherstellen, dass Ihre Klasse zum Muster von `animationSelector` passt

Debug-Schritte:

1. Das `<html>`-Element während der Navigation auf Klassen wie `is-changing`, `is-animating` prüfen
2. Die berechneten Styles der Übergangselemente kontrollieren

Problem: Skripte laufen nach Navigation nicht

Ursache:
JavaScript wird nur beim ersten Seitenaufruf ausgeführt.

Lösung:
Reinitialisierungscode in `reinitializeScripts` ergänzen oder auf das Event `swup:contentReplaced` hören.

Problem: Formulare werden nicht korrekt abgesendet

Ursache:
Swup verarbeitet standardmäßig keine Formularübermittlungen.

Lösungen:

1. `data-no-swup` zum Submit-Link/Button hinzufügen
2. Das **Swup Forms Plugin** installieren (zu den CDN-URLs hinzufügen und aktivieren)

Problem: Scroll-Position bei fixiertem Header

Lösung: `scrollOffset` anpassen:

```
scrollOffset: 80, // Höhe Ihres fixierten Headers in Pixeln
```

Performance-Optimierung

Empfohlene Produktionseinstellungen

```
const SWUP_CONFIG = {
  // Debug-Logging deaktivieren
  debug: false,

  // Caching optimieren
  cacheEnabled: true,
  maxCacheEntries: 30,
  cacheTTL: 10 * 60 * 1000, // 10 Minuten

  // Preloading aktivieren
  preloadEnabled: true,
  preloadOnHover: true,
  preloadThrottle: 5,

  // Nicht benötigte Plugins deaktivieren
  scriptsEnabled: false, // Nur aktivieren, wenn unbedingt nötig
  progressBarEnabled: true,

  // Native Transitions für unterstützte Browser aktivieren
  nativeViewTransitions: true,
};
```

Auswirkungen messen

Verwenden Sie die Browser-DevTools zum Vergleich von:

1. **Network Waterfall** – Voller Seitenload vs. Swup-Navigation
2. **Performance-Timeline** – Zeit bis zur Interaktivität messen
3. **Core Web Vitals** – LCP, FID, CLS vor und nach der Implementierung beobachten

---

Erweiterte Anpassung

Benutzerdefinierte Animation pro Link

Verwenden Sie das Attribut `data-swup-animation`:

```
<a href="/gallery" data-swup-animation="slide">Galerie ansehen</a>
```

Dies fügt während des Übergangs die Klasse `to-slide` zu `<html>` hinzu. Erstellen Sie dazu passendes CSS:

```
html.is-changing.to-slide .transition-fade {
  transition: transform 0.4s ease;
}

html.is-leaving.to-slide .transition-fade {
  transform: translateX(-100%);
}

html.is-rendering.to-slide .transition-fade {
  transform: translateX(100%);
}
```

Programmatische Navigation

Greifen Sie global auf die Swup-Instanz zu:

```
// Zu einer Seite navigieren
window.swup.navigate('/about');

// Ohne Animation navigieren
window.swup.navigate('/contact', { animate: false });

// Cache leeren
window.swup.cache.clear();
```

Bestimmte Seiten ausschließen

Seiten dynamisch anhand der URL ignorieren:

```
ignoreVisit: function(url, { el } = {}) {
  // Checkout-Seiten ignorieren
  if (url.includes('/checkout/')) return true;
```

```
  // Bestimmte Seite ignorieren
  if (url === '/special-page/') return true;

  // Element gegen Ignore-Selektoren prüfen (Standardverhalten)
  const { ignoreSelectors } = SWUP_CONFIG;
  if (el) {
    for (const selector of ignoreSelectors) {
      if (el.closest(selector)) return true;
    }
  }

  return false;
}
```

Dieser Leitfaden deckt die vollständige Einrichtung und Anpassung der SwupJS-Integration für Bricks Builder ab. Bei weiteren Fragen oder Problemen beziehen Sie sich auf die offizielle **Swup-Dokumentation** oder prüfen Sie die ausführlichen Kommentare in der Datei `swup-bricks-integration.js`.

Typografie in WP

wptexturize() in WordPress

Was es macht

`wptexturize()` ist eine WordPress-Kernfunktion (in `wp-includes/formatting.php`), die einfache, "dumme" Zeichen in typografisch korrekte Sonderzeichen umwandelt – ähnlich dem "Smart Quotes"-Feature aus Textverarbeitungsprogrammen.

Konkret ersetzt sie unter anderem:

Eingabe	Ausgabe	Bezeichnung
`'` (gerades Anführungszeichen)	`‘` / `’`	typografische Apostrophe/Anführungszeichen
`"`	`“` / `”`	typografische doppelte Anführungszeichen
`--`	`–`	En-Dash
`---`	`—`	Em-Dash
`...`	`…`	Ellipse (Auslassungspunkte)
`(c)`	`©`	Copyright-Zeichen
`(r)`	`®`	Registered-Trademark-Zeichen
`(tm)`	`™`	Trademark-Zeichen
`1/4`, `1/2` usw.	`¼`, `½`	Bruchzahlen *(teilweise)*
mehrere Bindestriche	entsprechende Gedankenstriche	

Zusätzlich schützt die Funktion bestimmte Bereiche vor der Umwandlung – z. B. Inhalte in `<pre>`, `<code>`, `<script>`, `<style>`-Tags sowie Shortcodes, damit dort keine ungewollten Zeichenersetzungen passieren.

Wo wird sie angewendet?

Standardmäßig hängt WordPress `wptexturize()` per Filter an mehrere Ausgabepunkte:

- `the_content`
- `the_title`

- `the_excerpt`
- Kommentartexte
- Widget-Texte
- u. v. m.

Das läuft über den `content_filters`-Mechanismus, meist mit Priorität `10`.

Kann man das steuern?

Ja, auf mehreren Ebenen:

1. Komplett deaktivieren

Für einen bestimmten Filter (z. B. Content) den Hook entfernen:

```
remove_filter( 'the_content', 'wptexturize' );
remove_filter( 'the_title', 'wptexturize' );
remove_filter( 'the_excerpt', 'wptexturize' );
```

Das muss nach der Filterregistrierung laufen, z. B. im `init`-Hook oder später:

```
add_action( 'init', function() {
    remove_filter( 'the_content', 'wptexturize' );
} );
```

2. Über den `run_wptexturize`-Filter global steuern

WordPress bietet einen dedizierten Filter, um die Funktion an- oder abzuschalten:

```
add_filter( 'run_wptexturize', '__return_false' );
```

⚠ Das deaktiviert die Funktion *global* für alle Aufrufe.

3. Ergebnis nachträglich anpassen

Über den Filter `wptexturize_output` lässt sich das Ergebnis nachbearbeiten – etwa um bestimmte Ersetzungen wieder zurückzudrehen:

```
add_filter( 'wptexturize_output', function( $output ) {
    // eigene Logik, z. B. bestimmte Ersetzungen rückgängig machen
    return $output;
} );
```

4. Für bestimmte Inhalte gezielt umgehen

Innerhalb von Content kann man Bereiche mit `<code>`, `<pre>` oder `<script>`/`<style>`-Tags schützen – dort greift `wptexturize()` ohnehin nicht.

5. Direkt im Theme/Plugin aufrufen oder umgehen

Man kann die Funktion auch manuell aufrufen:

```
echo wptexturize( $custom_text );
```

...falls man sie gezielt nur an einer Stelle nutzen möchte, ohne die globalen Hooks zu berühren.

Praktischer Tipp

Häufigster Anwendungsfall für das Deaktivieren: Wenn eigener Code, Shortcodes oder JSON-Ausgaben im Content stehen und durch die Zeichenersetzung (v. a. Anführungszeichen) kaputtgehen. Dann ist Variante **1** (gezieltes `remove_filter`) meist die sauberste Lösung, da sie andere Bereiche unangetastet lässt.

Falls du willst, kann ich dir auch zeigen, wie man das **nur für einen bestimmten Post-Typ oder Block** steuert – sag einfach Bescheid! □□

Plugin zum Steuern von wptexturize()

1. **wp-Typography** (empfehlenswert, sehr umfangreich)

Das ist das bekannteste Plugin für dieses Thema. Es **deaktiviert** `wptexturize()` **intern komplett** und ersetzt es durch eine eigene, viel granularere Engine. Im Backend kannst du z. B. einzeln steuern:

- Anführungszeichen (verschiedene Sprach-Stile, „deutsche“ vs. „englische“ Anführungszeichen)
- Gedankenstriche / Bindestriche (en-dash, em-dash)
- Ellipsen (…)
- geschützte Leerzeichen bei Einheiten, Initialen, Zahlen
- Ligaturen, Hochzahlen, Brüche
- Silbentrennung (Hyphenation) inkl. deutscher Wörterbücher
- Ausnahmen pro Tag/Klasse (z. B. `<code>` unberührt lassen)

Das ist die feinste Kontrolle, die es im Plugin-Bereich gibt – nicht nur "an/aus", sondern für jede einzelne Texturize-Funktion einstellbar.

□□ https://wordpress.org/plugins/wp-typography/

2. **Disable wptexturize** (einfache Ein/Aus-Lösung)

Ältere, sehr schlanke Plugins wie "*Disable wptexturize*" schalten die Funktion komplett ab – ohne granulare Einstellungen. Praktisch, wenn du **gar keine** Texturize-Transformation willst.

3. **Eigener Code-Snippet (kein Plugin, aber die "offizielle" Methode)**

WordPress bietet dafür selbst einen Filter-Hook an:

```
add_filter( 'run_wptexturize', '__return_false' );
```

Das deaktiviert die Funktion komplett und global. Willst du sie nur **teilweise** steuern (z. B. nur Anführungszeichen, aber keine Bindestriche), musst du in die internen Einstellungen von `wptexturize()` eingreifen – dafür gibt es aber keinen offiziellen Filter pro Einzelfunktion, weshalb genau hier **wp-Typography** seinen Mehrwert hat, weil es diese Granularität über die Oberfläche bereitstellt.

Kurzfassung / Empfehlung

Bedarf	Lösung
Nur komplett abschalten	"Disable wptexturize"-Plugin oder Code-Snippet
Feine Kontrolle (Anführungszeichen, Gedankenstriche, Einheiten etc. einzeln)	**wp-Typography**
Volle Codekontrolle ohne Plugin	`add_filter('run_wptexturize', '__return_false');` in `functions.php`

Wenn du sagst, was genau du steuern willst (z. B. nur die Anführungszeichen sollen unverändert bleiben, der Rest aber texturized werden), kann ich dir die passende Konfiguration bzw. den passenden Filter-Code dafür geben.

wptexturize() in BricksBuilder & Polylang

Kurze Antwort

Teilweise. Es kommt darauf an, welches Bricks-Element du verwendest:

Element	wptexturize() aktiv?
Rich Text Element	✅ Ja – läuft durch die WP-Content-Pipeline (inkl. `wpautop`, `wptexturize`)
Basic Text Element	❌ Nein – gibt den Text "roh" aus, ohne die klassischen Content-Filter
Heading, Text Link, etc.	❌ In der Regel nein, gleiche Ursache wie beim Basic Text
WP-Query/Post-Content via `{content}`-Dynamic-Data	✅ Ja, wenn wirklich `the_content` genutzt wird

Das wurde auch im offiziellen Bricks-Forum bestätigt: Nutzer beschwerten sich, dass Anführungszeichen im *Rich Text Element* automatisch "prettified" (→ typografische Anführungszeichen) werden, während das *Basic Text Element* unverändert bleibt. Die Bricks-Entwickler bestätigten: Das ist WP-Core-Verhalten (`wptexturize`), kein Bug in Bricks selbst.

Wer es abschalten will (nicht was du willst, aber zur Einordnung):

```
add_filter( 'run_wptexturize', '__return_false' );
```

Und wie ist es mit mehreren Sprachen (Polylang)?

`wptexturize()` holt sich die passenden „intelligenten" Anführungszeichen (z. B. „…" im Deutschen vs. “…” im Englischen vs. «…» im Französischen) aus **übersetzten Strings** im WordPress-Core (`wp-includes/formatting.php`), die über die geladene **.mo/.po-Datei der aktuell aktiven Locale**

kommen – nicht statisch für „Deutsch" hartcodiert.

Das heißt konkret:

1. **Wenn Polylang die Locale für die jeweils angezeigte Sprachversion korrekt setzt** (was es standardmäßig via `locale`-Filter frühzeitig im Request tut, bevor Content gerendert wird), dann bekommt `wptexturize()` beim Rendern der Seite auch die **richtige Locale** und liefert automatisch die passenden landestypischen Anführungszeichen – **aber nur für Elemente, die überhaupt durch `wptexturize()` laufen** (siehe Tabelle oben, also praktisch nur das Rich Text Element bzw. echten `the_content`-Output).
2. **Wichtige Einschränkung:** Die übersetzten Zeichen-Arrays innerhalb von `wptexturize()` werden **einmal pro Request statisch gecached** (`static` Variablen in der Funktion). Das ist in der Regel unproblematisch, weil eine Frontend-Anfrage ja nur eine Sprache/Locale betrifft – es sei denn du hast Setups, die mitten im Request die Sprache umschalten (z. B. bestimmte Multisite- oder Caching-Konstellationen), dann kann es zu falschen/veralteten Zeichen kommen.
3. **Basic Text / Heading Elemente bekommen NIE lokalisierte Anführungszeichen**, unabhängig von Polylang – weil `wptexturize()` dort gar nicht aufgerufen wird. Wenn du in diesen Feldern `"Text"` schreibst, bleibt es ein gerades Anführungszeichen, egal welche Sprache aktiv ist.

Praktische Empfehlung

- **Willst du korrekt lokalisierte Anführungszeichen** → nutze für Textinhalte mit Anführungszeichen das **Rich Text Element** statt Basic Text/Heading, dann greift `wptexturize()` + Polylang-Locale zusammen automatisch richtig.
- **Willst du es für Basic Text/Heading auch erzwingen**, kannst du selbst einen Filter auf die Bricks-Render-Ausgabe legen, z. B.:

```
add_filter( 'bricks/text-basic/render_content', function( $content ) {
    return wptexturize( $content );
}, 10, 1 );
```

(Hook-Name kann je Bricks-Version leicht variieren – ggf. `bricks/heading/output` oder `bricks/text/content` prüfen, da Bricks nicht für jedes Element denselben Filternamen bereitstellt.)

- **Alternative:** Direkt schon in der ACF/Text-Eingabe die richtigen Unicode-Zeichen „ “ ” , ' ' verwenden, statt dich auf `wptexturize()` zu verlassen – dann ist es unabhängig vom Element und von Polylang-Locale-Timing zuverlässig korrekt, gerade bei sprachübergreifenden Multilingual-Setups.

Fazit: `wptexturize()` funktioniert in Bricks nur dort, wo Bricks tatsächlich den klassischen WordPress-Content-Filter-Stack nutzt (Rich Text Element). Mit Polylang bekommst du dort auch

korrekt lokalisierte Anführungszeichen, sofern die Locale beim Rendern richtig gesetzt ist. Für alle anderen Bricks-Elemente (Basic Text, Heading etc.) greift die Funktion nicht – dort musst du selbst nachhelfen oder gleich die richtigen Zeichen eintippen.

Security

Security

wp-content und wp-includes zu schützen?

Kurze Antwort: Teilweise sinnvoll, aber mit wichtigen Einschränkungen

Es macht **eingeschränkt Sinn**, aber man muss unterscheiden, was genau "geschützt" wird – ein Pauschalblock kann die Seite auch kaputt machen.

wp-includes schützen meist unproblematisch

- `wp-includes` enthält WordPress-Core-Dateien (PHP-Klassen, JS, CSS), die **nicht direkt aufgerufen** werden müssen – sie werden von WordPress selbst intern geladen.
- Typische Regel (Apache/.htaccess):

```
RewriteRule ^wp-includes/[^/]+\.php$ - [F,L]
```

- **Nutzen:** Verhindert direktes Ausführen von PHP-Skripten in `wp-includes`, was ein gängiger Angriffsvektor ist (z.B. wenn eine Lücke eine Datei dort hochlädt/ausnutzt).
- **Risiko:** minimal – manche Plugins/Themes greifen selten direkt auf JS/CSS-Dateien in `wp-includes` zu (z.B. `wlwmanifest.xml`), daher sollte man PHP-Dateien blocken, nicht den ganzen Ordner inkl. Assets.

wp-content schützen ⚠ vorsichtiger, weil dort "lebende" Inhalte liegen

`wp-content` ist kein einheitlicher Ordner – er enthält:

Unterordner	Zweck	Direkter Zugriff nötig?
`uploads/`	Bilder, PDFs, Medien	**Ja**, muss öffentlich erreichbar sein
`themes/`	CSS/JS/Bilder des Themes	**Ja**, für Assets
`plugins/`	Plugin-Code, teils Assets	**Ja** für Assets, **Nein** für PHP direkt
`uploads/*.php`	Sollte es gar nicht geben	Wird oft als Malware-Ablage genutzt

Sinnvoll ist daher **nicht** "wp-content sperren", sondern gezielt:

```
# PHP-Ausführung in uploads verhindern (häufigster Angriffsweg)
<Directory "/wp-content/uploads/">
    <FilesMatch "\.php$">
        Require all denied
    </FilesMatch>
</Directory>
```

Das ist die **wichtigste Einzelmaßnahme** überhaupt, weil viele Malware-Uploads/Backdoors genau darauf abzielen, eine PHP-Datei in `uploads/` abzulegen und dann per Direktaufruf auszuführen. `uploads/` sollte generell **niemals PHP ausführen dürfen**.

Was FlyWP wahrscheinlich konkret macht

FlyWP (und ähnliche Managed-Hosting-Tools wie Kinsta, RunCloud, GridPane) bieten meist vorkonfigurierte Regeln für genau diese Kombination:

1. PHP-Ausführung in `wp-content/uploads` blocken
2. Direkten PHP-Zugriff auf `wp-includes` blocken
3. Directory Listing (Browsing) in beiden Ordnern deaktivieren
4. Zugriff auf sensible Dateien (`readme.html`, `wp-config.php`, `.git`, etc.) sperren

Das ist im Prinzip Standard-Hardening, wie es auch im offiziellen WordPress Hardening Guide empfohlen wird.

Fazit

- **Ja, aktivieren** – es ist ein Low-Risk/High-Value-Hardening-Schritt.
- Wichtig ist nur: **Assets (CSS/JS/Bilder) müssen weiter erreichbar bleiben**, nur die **PHP-Ausführung** in Upload-Verzeichnissen und der **direkte Aufruf von Core-PHP-Dateien** in `wp-includes` sollte blockiert werden.

- Danach kurz testen: Seite laden, Bilder/CSS/JS prüfen, ggf. Page-Builder/Plugin-Funktionen (manche legen z.B. generierte CSS-Dateien in `uploads/` ab – das bleibt unberührt, da nur PHP-Ausführung gesperrt wird, keine anderen Dateitypen).

Wenn du willst, kann ich dir konkrete .htaccess- oder Nginx-Regeln für deinen Stack zusammenstellen.

Performance

Performance

WordPress-Plugins für Server-Auslastung in der Adminleiste

Ja, es gibt einige Plugins, die genau das leisten – mit kleinen Einschränkungen, die für dich als Bricks/Divi-Anwender relevant sein könnten. □□

□□ Direkte Kandidaten

1. WP Server Health Stats

- Zeigt **CPU-Load, RAM-Nutzung, PHP-Memory, DB-Stats** und Server-Uptime an
- Aktualisiert sich per Ajax standardmäßig alle **200 ms** – also wirklich in Echtzeit
- Frei konfigurierbares Intervall & Farbschema
- Ergebnisse können in der Adminleiste angezeigt werden

Wichtiger Hinweis: Das Plugin nutzt `shell_exec()`, um an OS-Level-Daten zu kommen. Auf vielen **Managed-/Shared-Hosting-Umgebungen** (z. B. bei vielen Agentur-Hostern) ist diese PHP-Funktion aus Sicherheitsgründen deaktiviert – dann zeigt das Plugin nur Fehlercodes an. Außerdem gab es 2024 einen Vorfall, bei dem der Account des Entwicklers gehackt und Malware über eine Version des Plugins verteilt wurde (mittlerweile bereinigt, Version 1.8.0+). Ich würde es daher nur auf vertrauenswürdigen, root-eigenen Servern (VPS) einsetzen und aktuell halten.

2. MyServerInfo

- Zeigt **CPU Usage** *(Ø über 1 Minute)*, **RAM-Nutzung**, **Disk-Nutzung** sowie diverse PHP-/MySQL-Werte
- Über Checkboxen kannst du gezielt einzelne Metriken (`MEM: X%`, `AVG CPU: Y%`, `Disk: Z%`) in die Adminleiste holen
- Liest Werte aus `/proc/loadavg` bzw. `/proc/uptime` – funktioniert also nur auf **Linux-Servern**, nicht bei Windows-Hosting oder stark isolierten Containern

⚠ Grundsätzliche Einschränkung, die du kennen solltest

Da du (nach eigener Aussage) mit den internen WordPress-/PHP-Prozessen weniger vertraut bist, hier der Kontext, warum solche Plugins nicht überall gleich gut funktionieren:

```
flowchart TD
    A["PHP-Skript WordPress"] --> B{"Server erlaubt<br/>shell_exec / proc-Zugriff?"}
    B -->|Ja, z.B. eigener VPS| C["Echte Live-Werte<br/>CPU, RAM, Load"]
    B -->|Nein, z.B. Managed Hosting| D["Nur Näherungswerte<br/>oder Fehleranzeige"]
```

Viele hochwertige Managed-WordPress-Hoster (die für Bricks/Divi-Projekte oft empfehlenswert sind) sperren genau diese Low-Level-Funktionen aus Sicherheitsgründen. Das ist **kein Plugin-Fehler**, sondern Hosting-Policy.

□□ Für deinen tatsächlichen Anwendungsfall vermutlich nützlicher: Query Monitor

Wenn es dir eigentlich darum geht, **warum eine Seite mit Bricks oder Divi langsam lädt**, ist **Query Monitor** der Industriestandard und noch relevanter als reine Server-Metriken:

- Zeigt in der Adminleiste direkt: *Page Generation Time, Peak Memory Usage, Anzahl & Dauer der SQL-Queries*
- Klick auf die Leiste öffnet ein Panel mit genauen Angaben, **welcher Hook/welches Plugin** wie viel Zeit/Speicher verbraucht
- Läuft rein PHP-intern (kein `shell_exec` nötig) → funktioniert auf **jedem** Hosting
- Ideal, um z. B. herauszufinden, ob eine langsame Bricks-Seite an einem bestimmten Plugin, zu vielen DB-Queries oder einer schlecht optimierten Template-Struktur liegt

---

□□ Kurzübersicht

Plugin	Zeigt echte OS-Werte?	Funktioniert überall?	Fokus
WP Server Health Stats	Ja *(shell_exec)*	Nein – braucht Freigabe	Server-Hardware
MyServerInfo	Ja *(proc-Dateien)*	Nur Linux	Server-Hardware
Query Monitor	Nein *(PHP-intern)*	Ja	Seiten-Performance/Debugging

Empfehlung: Teste zuerst *MyServerInfo* für einen schnellen Blick auf die reine Server-Auslastung – und installiere zusätzlich *Query Monitor*, wenn du konkrete Performance-Probleme in Bricks- oder Divi-Projekten aufspüren willst. Beide zusammen ergänzen sich gut. □□