CS

Streamline Code Snippets Reference

Small custom PHP/CSS snippets used across the Streamline Diving site

6 Snippets | Reference Guide

Contents

1. Overview & How to Install a Snippet
   a. Installing a Snippet
2. Remove Title Product Count
3. Show Stock Count or Special-Order Message
4. Force Red Bold Text for Backorder Items in Cart
5. Display Zero-Total Message in Cart
6. Force Password Reset Form Without Session Cookies
7. Streamline POS — Frontend Access Snippet (v11)
   a. What's New in v11
   b. Installation
   c. How the URL Configuration Actually Works
   d. Finding the POS Plugin's Files
   e. What the Page Actually Loads
   f. The Iframe Token System
   g. Troubleshooting
8. Version History

1. Overview & How to Install a Snippet

These are small, standalone PHP/CSS snippets — not full plugins — that change specific behavior on the Streamline Diving WooCommerce site. Each one is installed using the free **Code Snippets** plugin (or WPCode), rather than being packaged as its own plugin.

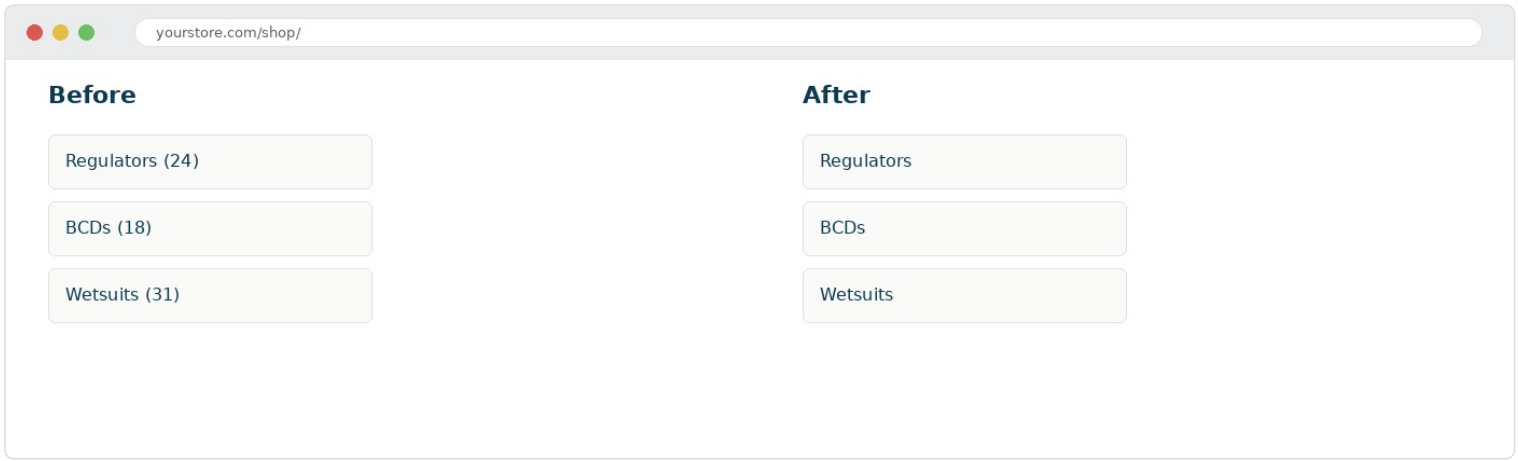
Installing a Snippet

1. Install and activate the **Code Snippets** plugin from the WordPress Plugin directory, if not already active.
2. Go to **Snippets → Add New**.
3. Give the snippet a descriptive name (matching the section titles below is a good convention).
4. Paste the code from the corresponding section into the code editor.
5. Set the snippet to run **everywhere** (unless otherwise noted) and save it as **Active**.

> **Note:** these snippets are lightweight, targeted fixes and customizations — each one documented here does exactly one job. If a snippet stops behaving as expected after a WooCommerce or theme update, check this reference first to confirm what it's supposed to do, then re-test.

2. Remove Title Product Count

Effect: hides the "(24)" product-count number that WooCommerce normally appends after each category name.



Before and after — the numeric count disappears from category names and widgets.

WooCommerce, by default, shows how many products are in a category — in the shop's category list, in category widgets, and in dropdown menus (e.g. "Regulators (24)"). This one-line snippet removes that count everywhere it would normally appear, for a cleaner look.

```
add_filter( 'woocommerce_subcategory_count_html', '__return_null' );
```

Tip: this only hides the displayed count — it doesn't change how many products are actually in each category, and it applies site-wide with no configuration needed.

3. Show Stock Count or Special-Order Message

Effect: shows the live stock number when a product is in stock, or an "Available - special order item" message (styled in red) when a stock-managed product's quantity is at zero.

This snippet has two parts:

- It overrides WooCommerce's default availability text so that any stock-managed product with quantity greater than zero shows the exact number in stock (e.g. "6 in stock") instead of a generic "In stock" label.
- For stock-managed products at zero quantity, it replaces the usual "Available on backorder" wording with "Available - special order item" — clearer phrasing for a dive shop where a backordered item usually means a special vendor order, not a delayed restock.
- It also overrides the backorder notification shown in the cart with the same red, bold "Available - special order item" text, so the message is consistent from the product page through to checkout.

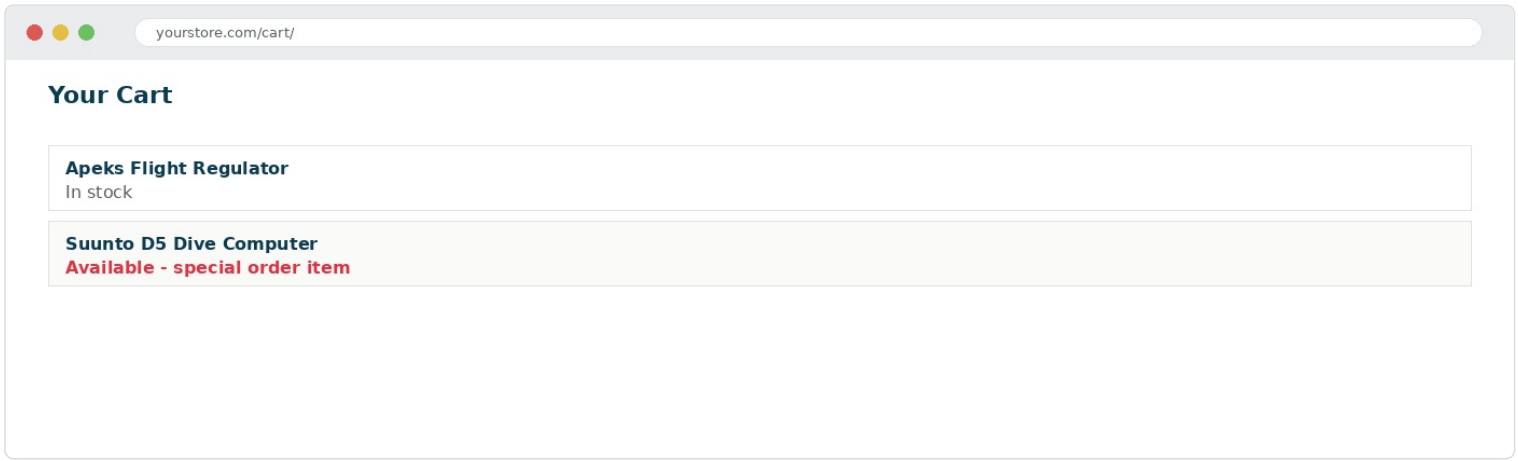
```
/**
 * Show stock count, or special order message if 0
 */
add_filter('woocommerce_get_availability', 'streamline_custom_availability', 10, 2);
function streamline_custom_availability($availability, $product) {
    if ($product->managing_stock()) {
        $stock_quantity = $product->get_stock_quantity();

        if ($stock_quantity > 0) {
            // Show stock count
            $availability['availability'] = sprintf(__('%s in stock', 'woocommerce'), $stock_quantity);
            $availability['class'] = 'in-stock';
        } else {
            // Show special order message
            $availability['availability'] = 'Available - special order item';
            $availability['class'] = 'available-on-backorder';
        }
    }
    return $availability;
}

/**
 * Replace backorder notification text
 */
add_filter('woocommerce_cart_item_backorder_notification', 'streamline_backorder_notification', 10, 2);
function streamline_backorder_notification($html, $product_id) {
    return '<p class="backorder_notification" style="color: #dc3232 !important; font-weight: 700 !important;">Available - special order
item</p>';
}
```

4. Force Red Bold Text for Backorder Items in Cart

Effect: pure CSS — forces the backorder status text in the cart to render in red, bold type, regardless of what theme styles might otherwise apply.



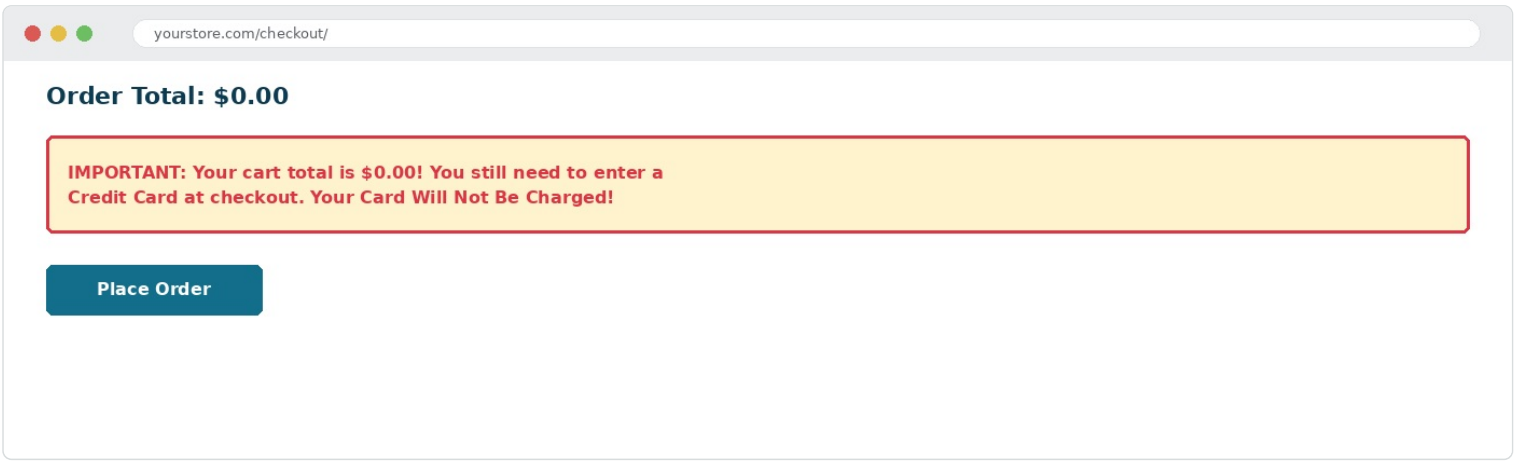
A backordered line item's status stands out in red, bold text against normal in-stock items.

This is a companion to the previous snippet — where that one controls the *text* shown for backorder items, this one guarantees the *styling* stays red and bold in the cart even if a theme update or plugin changes the default stock-status styling. It targets the cart page specifically, using `!important` to override conflicting theme CSS.

```
/**
 * Add CSS to force red bold text for backorder items in cart
 */
add_action('wp_head', 'streamline_backorder_cart_css', 999);
function streamline_backorder_cart_css() {
    ?>
    <style>
        .woocommerce-cart .stock.available-on-backorder,
        .woocommerce-cart-form .stock.available-on-backorder,
        .cart_item .stock.available-on-backorder {
            color: #dc3232 !important;
            font-weight: 700 !important;
        }
    </style>
    <?php
}
```

5. Display Zero-Total Message in Cart

Effect: shows a prominent red-bordered notice on the cart and checkout pages whenever the order total is $0.00 (from rewards points, gift cards, or a coupon), reminding the customer they still need to enter a card even though nothing will be charged.



The zero-total notice appears above the payment section whenever rewards, gift cards, or a coupon bring the order to $0.00.

Without this notice, a customer whose order comes to exactly $0.00 (commonly from applying enough reward points or a gift card balance) can be confused about why they're still asked to enter a credit card at checkout — WooCommerce still requires a valid payment method on file even when nothing will actually be charged. This snippet adds a clear explanation in three redundant ways, so it reliably shows up regardless of theme or page builder:

- **Method 1:** hooks directly into the checkout page, right before the payment section
- **Method 2:** hooks into the cart page, right before the "Proceed to Checkout" button
- **Method 3:** a JavaScript fallback that checks for a $0.00 total and inserts the same message near the payment area, in case the PHP hooks don't fire — for example, if Elementor or a similar page builder has replaced the standard cart/checkout templates

```
/*
 * Zero Total Checkout Message - Works with Rewards Points
 * Shows message when cart total is $0 due to rewards, gift cards, or coupons
 */

// Method 1: Add to checkout page (before payment section)
add_action('woocommerce_review_order_before_payment', 'show_zero_total_message_checkout', 10);

function show_zero_total_message_checkout() {
    // Check if cart total is $0
    if (WC()->cart && WC()->cart->get_total('raw') == 0) {
        echo '<div class="zero-total-notice" style="
            background: #fff3cd;
            border: 3px solid #dc3545;
            padding: 20px;
            margin: 20px 0;
            border-radius: 5px;
            text-align: center;
            box-shadow: 0 2px 5px rgba(0,0,0,0.1);
        ">';
        echo '<p style="
            margin: 0;
            color: #dc3545;
            font-size: 18px;
            line-height: 1.8;
        ">';
        echo 'IMPORTANT: Your cart total is $0.00! You still need to enter a Credit Card at checkout. Your Card Will Not Be Charged!';
        echo '</p>';
        echo '</div>';
    }
}

// Method 2: Also add to cart page (before checkout button)
add_action('woocommerce_proceed_to_checkout', 'show_zero_total_message_cart', 5);

function show_zero_total_message_cart() {
    // Check if cart total is $0
    if (WC()->cart && WC()->cart->get_total('raw') == 0) {
        echo '<div class="zero-total-notice-cart" style="
            background: #fff3cd;
            border: 3px solid #dc3545;
            padding: 20px;
            margin: 20px 0;
            border-radius: 5px;
            text-align: center;
            box-shadow: 0 2px 5px rgba(0,0,0,0.1);
        ">';
        echo '<p style="
```

```
            margin: 0;
            color: #dc3545;
            font-size: 18px;
            line-height: 1.8;
        ">';
        echo 'IMPORTANT: Your cart total is $0.00! You still need to enter a Credit Card at checkout. Your Card Will Not Be Charged!';
        echo '</p>';
        echo '</div>';
    }
}

// Method 3: JavaScript fallback (if hooks don't work due to Elementor)
add_action('wp_footer', 'zero_total_javascript_fallback');

function zero_total_javascript_fallback() {
    if (!is_checkout() && !is_cart()) {
        return;
    }

    // Only run if total is $0
    if (WC()->cart && WC()->cart->get_total('raw') != 0) {
        return;
    }
    ?>
    <script>
    jQuery(document).ready(function($) {
        // Check if message already exists from PHP
        if ($('.zero-total-notice').length > 0 || $('.zero-total-notice-cart').length > 0) {
            console.log('Zero total message already inserted via PHP hook');
            return;
        }

        console.log('Zero total detected - inserting message via JavaScript');

        var message = '<div class="zero-total-notice-js" style="background: #fff3cd; border: 3px solid #dc3545; padding: 20px; margin: 20px 0;
border-radius: 5px; text-align: center; box-shadow: 0 2px 5px rgba(0,0,0,0.1);">' +
            '<p style="margin: 0; color: #dc3545; font-size: 18px; line-height: 1.8;">' +
            'IMPORTANT: Your cart total is $0.00! You still need to enter a Credit Card at checkout. Your Card Will Not Be Charged!' +
            '</p></div>';

        // Try multiple insertion points
        if ($('#place_order').length) {
            $(message).insertBefore('#place_order');
            console.log('Message inserted before #place_order');
        } else if ($('.woocommerce-checkout-payment').length) {
            $(message).prependTo('.woocommerce-checkout-payment');
            console.log('Message inserted in .woocommerce-checkout-payment');
        } else if ($('#payment').length) {
            $(message).prependTo('#payment');
            console.log('Message inserted in #payment');
        } else if ($('.wc_payment_methods').length) {
            $(message).insertBefore('.wc_payment_methods');
            console.log('Message inserted before .wc_payment_methods');
        } else if ($('form.checkout').length) {
            $(message).appendTo('form.checkout');
            console.log('Message appended to checkout form');
        } else {
            console.log('Could not find suitable insertion point');
        }
    });
    </script>
    <?php
}
```

6. Force Password Reset Form Without Session Cookies

Effect: ensures the "reset your password" form actually displays when a customer clicks the reset link in their email, even in hosting/caching setups where WooCommerce's normal session-cookie check fails to recognize the visit as a valid reset request.

WooCommerce's password reset flow normally depends on a browser session to confirm that a visit to the account page with a `?key=...&login=...` link is a legitimate reset request rather than someone guessing at the URL. On some hosts — particularly aggressive page caching or environments where PHP sessions don't start reliably — that check can fail silently, and the customer just sees the normal login form instead of the password reset fields.

This snippet works around that in four ways:

- Starts a PHP session on `init` if one hasn't already started
- Forces the `show-reset-form` flag on whenever a `key` and `login` parameter are both present in the URL
- Ensures a WooCommerce session object exists so the reset flow has somewhere to store its state
- Forces the query var that tells WooCommerce's template to render the reset fields rather than the login form

```
// Comprehensive password reset fix
add_action('init', function() {
    // Start session if not already started
    if (!session_id() && !headers_sent()) {
        session_start();
    }
});

// Force password reset form to display
add_action('template_redirect', function() {
    if (is_account_page() && isset($_GET['key']) && isset($_GET['login'])) {
        // Force the form to display
        $_GET['show-reset-form'] = 'true';
        $_REQUEST['show-reset-form'] = 'true';
    }
});

// Initialize WooCommerce session for password reset
add_filter('woocommerce_lost_password_url', function($url) {
    if (isset($_GET['key']) && isset($_GET['login'])) {
        // Ensure WooCommerce session exists
        if (function_exists('WC') && !WC()->session) {
            WC()->session = new WC_Session_Handler();
            WC()->session->init();
        }
    }
    return $url;
});

// Force display of reset password form
add_action('woocommerce_before_lost_password_form', function() {
    if (isset($_GET['key']) && isset($_GET['login'])) {
        // Force display the reset password fields
        global $wp;
        $wp->query_vars['show-reset-form'] = true;
    }
});
```

Note: if password resets still don't display correctly after adding this snippet, check whether a caching plugin is serving a fully cached version of the account page to logged-out visitors — password reset links should always be excluded from full-page caching.

7. Streamline POS — Frontend Access Snippet (v11)

Effect: serves the Streamline POS interface directly at a custom front-end URL (or several), without staff needing to log into WordPress at all — this is the code referenced in the Streamline POS manual's "Frontend Access" section.

> **No WordPress login required, by design:** this snippet has no permission check of any kind — no capability requirement, no redirect to `wp-login.php`. Visiting the URL always loads the POS's own login screen, and the real security gate is a POS account (created in the POS app under Settings → Users), completely separate from WordPress usernames and passwords. This is intentional — it's what makes the URL usable on a dedicated terminal that should never need a staff member signed into WordPress itself.

Version 11 is a significant upgrade over earlier releases of this snippet: it now supports **multiple POS URLs**, each with its own **default register** pre-selected. This is aimed at shops running more than one physical register or checkout station — for example, a main counter and a rental desk — where each station should open straight into its own register rather than requiring a manual switch after loading.

What's New in v11

- Reads a list of configured URLs (`posUrls`) from the POS app's own settings, each with a `slug` and an optional `registerName` — configured in-app under **POS app → Settings → General → POS Access URL**, not in the snippet itself
- Registers a WordPress rewrite rule for every configured URL automatically, and re-flushes rewrite rules whenever the set of URLs or registers changes — no manual visit to Settings → Permalinks required
- Automatically upgrades sites still using the older single-URL setting (`posUrlSlug`) from before multiple URLs were supported
- Carries the register name straight through the URL rewrite to the page loader, so a specific station's URL opens directly into its assigned register
- Includes the same secure iframe auto-login token system as earlier versions, for embedding the POS in an iframe with a pre-authenticated WordPress session (see "The Iframe Token System," below — this is separate from, and unrelated to, the no-login behavior of the direct front-end URL above)

Installation

1. Replace the existing Frontend Access snippet's code with this v11 version in Code Snippets.
2. Save and Activate.
3. In the POS app itself, go to **Settings → General → POS Access URL** and set your desired URL slug(s). If you run more than one register, give each URL its own default register there.
4. The URL(s) go live immediately — there's no need to visit Settings → Permalinks manually.

How the URL Configuration Actually Works

`streamline_pos_get_url_configs()` reads directly from the `wp_streamline_pos_settings` database table (the same table the POS app itself writes to), decodes the `posUrls` JSON, and sanitizes every slug through WordPress's own `sanitize_title()`. If nothing is configured yet, or the table doesn't exist, it falls back to a single unnamed `pos` URL with no default register:

```
function streamline_pos_get_url_configs() {
    global $wpdb;
    $table = $wpdb->prefix . 'streamline_pos_settings';
    $fallback = array( array( 'slug' => 'pos', 'registerName' => '' ) );

    if ( $wpdb->get_var( $wpdb->prepare( 'SHOW TABLES LIKE %s', $table ) ) !== $table ) {
        return $fallback;
    }
    $row = $wpdb->get_var( $wpdb->prepare(
        "SELECT setting_value FROM {$table} WHERE setting_key = %s",
        'general'
    ) );
    if ( ! $row ) { return $fallback; }
    $general = json_decode( $row, true );
    if ( ! empty( $general['posUrls'] ) && is_array( $general['posUrls'] ) ) {
        // ...builds one { slug, registerName } entry per configured URL
    }
    // Legacy single-URL setting, from before multiple URLs were supported
    if ( ! empty( $general['posUrlSlug'] ) ) {
        return array( array( 'slug' => sanitize_title( trim( $general['posUrlSlug'], '/' ) ), 'registerName' => '' ) );
    }
    return $fallback;
}
```

Each configured URL becomes its own WordPress rewrite rule, with the register name (if any) carried straight through the URL itself as a query var — so the page loader knows which register to pre-select without a second database lookup:

```
add_rewrite_rule(
    '^' . preg_quote( $slug, '/' ) . '/?$',
    'index.php?streamline_pos_page=1&streamline_pos_register=' . $register_param,
    'top'
);
```

Rewrite rules only get flushed when something actually changed — the snippet hashes the whole URL configuration and compares it against the hash from the last flush, so adding, removing, or re-pointing a URL triggers exactly one flush, and normal page loads don't pay that cost every time:

```
$config_hash  = md5( wp_json_encode( $configs ) );
$flushed_hash = get_option( 'streamline_pos_flushed_config_hash', '' );
if ( $flushed_hash !== $config_hash ) {
    update_option( 'streamline_pos_flushed_config_hash', $config_hash );
    flush_rewrite_rules( false );
}
```

Finding the POS Plugin's Files

Before serving the page, the snippet needs to locate the actual Streamline POS plugin so it can load its compiled CSS. It first checks for the plugin's own `STREAMLINE_POS_PLUGIN_URL` constant (the most reliable option, present on any reasonably current install); if that's not defined, it falls back to checking two known folder names, `sl-pos` and `streamline-pos`, for backward compatibility with older installs. If neither is found, it stops immediately with a clear message rather than serving a broken page: "*SL POS plugin not found. Please make sure it is installed and activated.*"

What the Page Actually Loads

The served page is a minimal HTML shell — not a copy of anything from wp-admin:

- A full-screen loading spinner while everything initializes
- Tailwind CSS loaded from its CDN, plus the POS plugin's own compiled stylesheet
- React 18 and ReactDOM loaded from unpkg's CDN
- A small inline script that builds `window.streamlinePosData` — the REST URL, a WordPress nonce, the site URL, the POS's own session token, the iframe auto-login token (see below), and an `isWpLoggedIn` flag — all of which the POS app's own JavaScript then reads to initialize itself

Nothing here checks whether the visitor is allowed to be on the page. `isWpLoggedIn` is informational only — it's passed along so the POS app can decide things like whether to offer a WordPress-session-based shortcut, but its absence does not block the page from loading, and its presence does not skip the POS's own login screen.

The Iframe Token System

This is a separate mechanism from the direct front-end URL above, used when the POS is embedded as an iframe inside another page in wp-admin — for example, inside one of the SL Extensions tabs described in the Streamline POS manual, Section 24. Because that embedding happens inside the WordPress admin area, it generates a short-lived, signed token tied to a specific **WordPress** user (by default, WordPress user ID 1 — typically the site's original administrator account) so the iframe doesn't need to prompt for a second login on top of the WordPress session that's already active there:

```
function streamline_pos_generate_iframe_token() {
    $user_id = streamline_pos_get_iframe_user_id(); // defaults to WP user ID 1
    $expires = time() + 3600; // valid for 1 hour
    $secret  = get_option( 'streamline_pos_session_token', '' );
    $payload = $user_id . '|' . $expires;
    $hmac    = hash_hmac( 'sha256', $payload, $secret );
    return rtrim( strtr( base64_encode( $payload . '|' . $hmac ), '+/', '-_' ), '=' );
}
```

The token is a base64-encoded payload of the user ID and an expiry timestamp, signed with HMAC-SHA256 using a per-site secret generated once and stored in `streamline_pos_session_token`. Validation re-computes the HMAC and uses a constant-time comparison (`hash_equals`) to prevent timing attacks, and rejects anything expired. This token only matters for the iframe-embedding scenario — it plays no role in, and does not bypass, the POS's own login screen when someone visits a Frontend Access URL directly.

Troubleshooting

Issue	What to check
New URL 404s after adding it in POS Settings	Give it a few seconds — the snippet auto-flushes rewrite rules when the URL configuration changes; a hard refresh usually resolves it.
Wrong register loads at a given URL	Confirm that URL's Default Register is set correctly under POS app → Settings → General → POS Access URL.
POS won't load at all — "plugin not found" message	Confirm the Streamline POS plugin itself is installed and active, under one of the expected folder names (sl-pos or streamline-pos) if it's an older/manual install.
The page loads but the login screen rejects every password	This is the POS's own login, not a WordPress one — confirm the username and password against a real POS account (created via the POS app's own Settings → Users), not a WordPress account.

For the complete script — including the full rewrite-rule registration and the page template shown above — see the file `streamline-pos-frontend-snippet.php` that shipped alongside this reference, or the *Frontend Access* section of the Streamline POS manual for the end-user-facing setup steps.

8. Version History

What's changed in this reference guide, in plain language — for reference and troubleshooting. Unlike the Streamline plugins, these are standalone code snippets with no plugin version number of their own, so this table tracks changes to the guide and its snippets directly, dated to when each change was actually made.

Date	What Changed
Sep 2026	Added this Version History section. Re-verified all 6 documented snippets (Sections 2–7) against their current source code — no changes found in any of them, including the Streamline POS Frontend Access snippet, which remains at v11 as documented.

About this table: this reference guide predates this Version History section, so earlier changes to individual snippets (such as when the POS Frontend Access snippet was first updated to v11) aren't individually dated here — only verifiable going forward. Every future change to a snippet or this guide gets a dated row here.