```
// Compiled Plugin Code
// Timestamp: 2025-09-12 06:49:15

// File: C:\laragon\www\multisite\wp-content\plugins\decentralized-user-
auth\admin\DuaAddUserPageController.php
<?php
namespace Dua\Admin;

// Prevent direct access.
defined('ABSPATH') || exit;

/**
 * Handles admin behaviors on the Add User page.
 * Used to suppress UI sections and block existing user additions.
 */
class DuaAddUserPageController {

    /**
     * Registers hooks for Add User page behaviors.
     * Hooked during plugin initialization.
     */
    public function __construct() {
        // Disable "Add Existing User" form on multisite user-new.php.
        add_filter('show_network_site_users_add_existing_form',
'__return_false');

        // Remove "Add Existing User" form on subsites user-new.php.
        add_action('admin_head', [$this, 'suppressExistingUserSection']);

        // Prevent existing users from being added to a site.
        add_filter('can_add_user_to_blog', [$this,
'preventExistingUserAddition'], 10, 4);
    }

    /**
     * Suppresses the "Add Existing User" section via CSS and JavaScript.
     * Only runs on site-level user-new.php page.
     *
     * @return void
     */
    public function suppressExistingUserSection() {
        global $pagenow;

        if ($pagenow !== 'user-new.php' || is_network_admin()) {
            return;
        }

        // Hide the existing form via CSS — safe and non-invasive.
        echo '<style> #add-existing-user, #add-existing-user + p,
#adduser { display: none; } </style>';

        // Remove the existing form via JavaScript — safe and non-
invasive.
        echo '<script>
            document.addEventListener("DOMContentLoaded", function () {
                const heading = document.getElementById("add-existing-
user");
                const paragraph = heading?.nextElementSibling;
                const form = document.getElementById("adduser");

                if (heading) heading.remove();
                if (paragraph && paragraph.tagName === "P")
paragraph.remove();
                if (form) form.remove();
            });
        </script>';
    }

    /**
     * Prevents existing users from being added to a site.
     * Returns WP_Error to block the action before it occurs.
     *
     * @param true|WP_Error $retval  Default true.
     * @param int           $user_id User ID.
     * @param string        $role    Role being assigned.
     * @param int           $blog_id Site ID.
     * @return true|WP_Error
     */
    public function preventExistingUserAddition($retval, $user_id, $role,
$blog_id) {
        if (get_current_blog_id() === $blog_id) {
            return new \WP_Error(
                'dua_existing_user_blocked',
                __('Adding existing users to a subsite is not allowed.',
'decentralized-user-auth')
            );
        }

        return $retval;
    }
}


// File: C:\laragon\www\multisite\wp-content\plugins\decentralized-user-
auth\admin\DuaLinkedAccountsProfile.php
<?php
namespace Dua\Admin;

use Dua\DuaAuthToken;

// Prevent direct access.
defined('ABSPATH') || exit;

/**
 * Renders the linked accounts UI on user profile pages.
 * Displays connected subsite accounts and provides linking interface.
 */
class DuaLinkedAccountsProfile {

    /**
     * Registers hooks to render UI on user profile pages.
     * Hooked into 'show_user_profile' and 'edit_user_profile'.
     */
    public function __construct() {
        add_action('show_user_profile', [$this, 'renderUi']);
        add_action('edit_user_profile', [$this, 'renderUi']);
    }

    /**
     * Renders the linked accounts interface in the user profile.
     * Only visible on the main site and for authorized users.
     *
     * @param \WP_User $user The user object being edited.
     * @return void
     */
    public function renderUi($user) {
        // Only render on the main site.
        if (get_current_blog_id() !== 1) {
            return;
        }

        // Only allow self-editing or super admin.
        if (!current_user_can('edit_user', $user->ID)) {
            return;
        }

        if (get_current_user_id() !== $user->ID && !is_super_admin()) {
            return;
        }

        $linkedAccounts = DuaAuthToken::getLinkedAccounts($user->ID);
        ?>

        <h2><?php esc_html_e('Connected Accounts', 'decentralized-user-
auth'); ?></h2>
        <?php wp_nonce_field('dua_link_account', 'dua_nonce'); ?>
        <input type="hidden" id="dua_main_user_id" value="<?php echo
esc_attr($user->ID); ?>">

        <table class="widefat fixed striped">
            <thead>
                <tr>
                    <th><?php esc_html_e('Site URL', 'decentralized-user-
auth'); ?></th>
                    <th><?php esc_html_e('Username', 'decentralized-user-
auth'); ?></th>
                    <th><?php esc_html_e('Email', 'decentralized-user-
auth'); ?></th>
                    <th><?php esc_html_e('Actions', 'decentralized-user-
auth'); ?></th>
                </tr>
            </thead>
            <tbody id="dua-linked-list">
                <?php if ($linkedAccounts): ?>
                    <?php foreach ($linkedAccounts as $account):
                        $token     = DuaAuthToken::generate($account->ID,
$account->site_id);
                        $loginUrl  = get_site_url($account->site_id) .
'/wp-login.php?action=remote_login&token=' . urlencode($token);
                    ?>
                        <tr>
                            <td><?php echo esc_url(get_site_url($account-
>site_id)); ?></td>
                            <td><?php echo esc_html($account-
>user_login); ?></td>
                            <td><?php echo esc_html($account-
>user_email); ?></td>
                            <td>
                                <a href="<?php echo esc_url($loginUrl);
?>" class="button button-secondary" target="_blank"><?php
esc_html_e('Sign In', 'decentralized-user-auth'); ?></a>
                                <button class="button button-link-delete
dua-unlink-user-account" data-user-id="<?php echo esc_attr($account->ID);
?>"><?php esc_html_e('Unlink', 'decentralized-user-auth'); ?></button>
                            </td>
                        </tr>
                    <?php endforeach; ?>
                <?php else: ?>
                    <tr id="no-linked-account">
                        <td colspan="4"><em><?php esc_html_e('No accounts
linked yet.', 'decentralized-user-auth'); ?></em></td>
                    </tr>
                <?php endif; ?>
            </tbody>
        </table><br>

        <h2><?php esc_html_e('Connect an Account', 'decentralized-user-
auth'); ?></h2>
        <table class="form-table link-account-fields-table" id="link-
account-fields-table">
            <tr>
                <th><label for="dua_site_url"><?php esc_html_e('Subsite
URL', 'decentralized-user-auth'); ?></label></th>
                <td><input type="url" id="dua_site_url" class="regular-
text" placeholder="https://example.site" required></td>
            </tr>
            <tr>
                <th><label for="dua_username"><?php esc_html_e('Username
or Email', 'decentralized-user-auth'); ?></label></th>
                <td><input type="text" id="dua_username" class="regular-
text" required></td>
            </tr>
            <tr>
                <th><label for="dua_password"><?php
esc_html_e('Password', 'decentralized-user-auth'); ?></label></th>
                <td><input type="password" id="dua_password"
class="regular-text" required></td>
            </tr>
            <tr>
                <th></th>
                <td>
                    <button id="dua-connect-button" type="button"
class="button button-secondary"><?php esc_html_e('Link Account',
'decentralized-user-auth'); ?></button>
                    <p id="dua-link-status" style="margin-top:8px;"></p>
                </td>
            </tr>
        </table>

        <?php
    }
}


// File: C:\laragon\www\multisite\wp-content\plugins\decentralized-user-
auth\admin\DuaNetworkSettings.php
<?php
namespace Dua\Admin;

use RecursiveIteratorIterator;
use RecursiveDirectoryIterator;
use FilesystemIterator;
use Dompdf\Dompdf;

// Prevent direct access.
defined('ABSPATH') || exit;

/**
```

```
 * Renders and manages network-wide plugin settings.
 * Includes cache, token, rate limit configuration and plugin code
compilation.
 */
class DuaNetworkSettings {

    /**
     * Min/max ranges (in seconds) for plugin settings.
     * Used for input constraints and validation.
     */
    protected $settingRanges = [
        'cache_expiry' => ['min' => 3600, 'max' => 86400],          //
1h–24h
        'roaming_cookie_expiry' => ['min' => 1800, 'max' => 43200], //
30m–12h
        'remote_login_token_expiry' => ['min' => 30, 'max' => 300], //
30s–5m
        'rate_limit_max' => ['min' => 3, 'max' => 10],              //
3s–10s
        'rate_limit_wait' => ['min' => 60, 'max' => 3600],          //
1m–1h
    ];

    /**
     * Registers hooks for rendering and saving network settings.
     * Hooked during plugin initialization.
     */
    public function __construct() {
        // Inject settings UI into network admin.
        add_action('wpmu_options', [$this, 'renderSettingsSection']);

        // Handle settings form submission.
        add_action('update_wpmu_options', [$this, 'saveSettings']);

        // Intercept settings page load to handle compile action.
        add_action('load-settings.php', [$this, 'handleCompileAction']);

        // Display admin notices based on query params.
        add_action('network_admin_notices', [$this,
'renderAdminNotices']);
    }

    /**
     * Renders the plugin settings section in network admin.
     * Hooked into 'wpmu_options'.
     */
    public function renderSettingsSection() {
        // Fetch current values for each setting.
        $roaming_secret_key = dua_get_roaming_secret_key();
        $is_default_roaming_secret_key = ($roaming_secret_key === 'dua-
super-consistent-network-secret');

        $fields = [
            'cache_expiry' => ['label' => 'Cache Expiry', 'value' =>
dua_get_cache_expiry()],
            'roaming_cookie_expiry' => ['label' => 'Roaming Cookie
Expiry', 'value' => dua_get_roaming_cookie_expiry()],
            'remote_login_token_expiry' => ['label' => 'Remote Login
Token Expiry', 'value' => dua_get_remote_login_token_expiry()],
            'rate_limit_max' => ['label' => 'Max Login Attempts', 'value'
=> dua_get_rate_limit_max()],
            'rate_limit_wait' => ['label' => 'Rate Limit Wait Time',
'value' => dua_get_rate_limit_wait()],
        ];

        echo '<h2 id="dua-settings">' . esc_html__('Decentralized User 
Authentication', 'decentralized-user-auth') . '</h2>';
        echo '<table class="form-table">';

        // Render each setting as a numeric input field.
        foreach ($fields as $key => $field) {
            $range = $this->settingRanges[$key];
            ?>
            <tr>
                <th scope="row">
                    <label for="dua_<?php echo esc_attr($key); ?>">
                        <?php echo esc_html($field['label']); ?>
                    </label>
                </th>
                <td>
                    <input type="number"
                           name="dua_<?php echo esc_attr($key); ?>"
                           id="dua_<?php echo esc_attr($key); ?>"
                           value="<?php echo esc_attr($field['value']); 
?>"
                           min="<?php echo esc_attr($range['min']); ?>"
                           max="<?php echo esc_attr($range['max']); ?>" 
/>
                    <span><?php echo esc_html__('Seconds', 
'decentralized-user-auth'); ?></span>
                    <p class="description">
                        <?php
                        // Show validation range as helper text.
                        printf(
                            esc_html__('Must be between %1$d seconds and 
%2$d seconds.', 'decentralized-user-auth'),
                            esc_html($range['min']),
                            esc_html($range['max'])
                        );
                        ?>
                    </p>
                </td>
            </tr>
            <?php
        }
        ?>

        <!-- Render the roaming secret key as a separate row. -->
        <tr>
            <th scope="row">
                <label for="dua_roaming_secret_key">
                    <?php esc_html_e('Roaming Secret Key', 
'decentralized-user-auth'); ?>
                </label>
            </th>
            <td>
                <input type="text"
                       name="dua_roaming_secret_key"
                       id="dua_roaming_secret_key"
                       value="<?php echo esc_attr($roaming_secret_key); 
?>"
                       class="regular-text"
                       readonly />
                <p class="description">
```

```
                    <?php esc_html_e('Used to sign roaming cookies across
subsites. Keep this secret.', 'decentralized-user-auth'); ?>
                </p>

                <?php if ($is_default_roaming_secret_key): ?>
                    <button type="button" class="button" id="dua-
generate-secret-key">
                        <?php esc_html_e('Generate New Key',
'decentralized-user-auth'); ?>
                    </button>

                    <p class="description" style="color: #d63638;">
                        <?php esc_html_e('You are using the default key.
Please generate a new one. You have to relogin after saving the new
key.', 'decentralized-user-auth'); ?>
                    </p>
                <?php endif; ?>
            </td>
        </tr>

        <!-- Render compile button as a separate row. -->
        <tr>
            <th scope="row"><label><?php esc_html_e('Compile Plugin
Code', 'decentralized-user-auth'); ?></label></th>
            <td>
                <button id="dua-compile-code-button"
                        type="button"
                        class="button button-secondary"
                        data-redirect="<?php echo
esc_url(network_admin_url('settings.php?action=compile-code')); ?>">
                    <?php esc_html_e('Compile', 'decentralized-user-
auth'); ?>
                </button>
            </td>
        </tr>
        <?php
        echo '</table>';
    }

    /**
     * Saves plugin settings submitted from the network admin form.
     * Validates input and updates site options.
     */
    public function saveSettings() {
        // Verify nonce for security.
        if (!check_admin_referer('siteoptions')) {

wp_redirect(network_admin_url('settings.php?error=security'));
            exit;
        }

        $keys = array_keys($this->settingRanges);
        $final = [];

        // Validate each submitted value against its range.
        foreach ($keys as $key) {
            $raw = absint($_POST['dua_' . $key] ?? 0);
            $range = $this->settingRanges[$key];

            if ($raw < $range['min'] || $raw > $range['max']) {

wp_redirect(network_admin_url('settings.php?error=invalid_input'));
```

```
                exit;
            }

            $final[$key] = $raw;
        }

        // Save validated values and clear related transients.
        foreach ($final as $key => $value) {
            update_site_option('dua_' . $key, $value);
            delete_transient('dua_' . $key . '_cached');
        }

        // Save roaming secret key
        if (isset($_POST['dua_roaming_secret_key'])) {
            error_log('DUA Roaming Secret Key Posted.');
            $key = sanitize_text_field($_POST['dua_roaming_secret_key']);
            update_site_option('dua_roaming_secret_key', $key);
            delete_transient('dua_roaming_secret_key_cached');
        }

        wp_redirect(network_admin_url('settings.php?updated=true'));
        exit;
    }

    /**
     * Renders admin notices based on query parameters.
     * Displays success or error messages after actions.
     */
    public function renderAdminNotices() {
        // Handle error messages.
        if (isset($_GET['error'])) {
            switch ($_GET['error']) {
                case 'security':
                    echo '<div class="notice notice-error"><p>' .
esc_html__('Security check failed. Please try again.', 'decentralized-
user-auth') . '</p></div>';
                    break;
                case 'unauthorized':
                    echo '<div class="notice notice-error"><p>' .
esc_html__('You are not authorized to compile plugin code.',
'decentralized-user-auth') . '</p></div>';
                    break;
                case 'invalid_input':
                    echo '<div class="notice notice-error"><p>' .
esc_html__('Invalid settings submitted. Please check your values in the
', 'decentralized-user-auth') . ' <a href="' .
esc_url(network_admin_url('settings.php#dua-settings')) . '">' .
esc_html__('Decentralized User Authentication Section', 'decentralized-
user-auth') . '</a>.</p></div>';
                    break;
            }
        }

        // Handle success message after compilation.
        if (isset($_GET['compiled'])) {
            echo '<div class="notice notice-success"><p>' .
esc_html__('Plugin code compiled successfully.', 'decentralized-user-
auth') . '</p></div>';
        }
    }

    /**
```

```
     * Handles the compile-code action triggered from the settings page.
     * Validates permission and triggers plugin code compilation.
     */
    public function handleCompileAction() {
        // Only proceed if in network admin and action matches.
        if (!is_network_admin() || ($_GET['action'] ?? '') !== 'compile-
code') {
            return;
        }

        // Check user capability before compiling.
        if (!current_user_can('manage_network')) {

wp_redirect(network_admin_url('settings.php?error=unauthorized'));
            exit;
        }

        // Run compilation and redirect with success flag.
        self::compilePluginCode();
        wp_redirect(network_admin_url('settings.php?compiled=true'));
        exit;
    }

    /**
     * Compiles all plugin PHP files into a single debug file.
     * Excludes vendor and misc directories. Outputs to TXT or PDF.
     */
    public static function compilePluginCode() {
        $basePath      = DUA_PLUGIN_DIR;
        $excludedDirs  = [realpath($basePath . '/z-misc'),
realpath($basePath . '/vendor')];
        $timestamp     = date('Y-m-d H:i:s');
        $outputFormats = ['txt' => false, 'pdf' => true];

        // Initialize compiled output with header.
        $compiledText = "// Compiled Plugin Code\n// Timestamp:
{$timestamp}\n\n";

        // Recursively scan plugin directory for PHP files.
        $iterator = new RecursiveIteratorIterator(
            new RecursiveDirectoryIterator($basePath,
FilesystemIterator::SKIP_DOTS)
        );

        foreach ($iterator as $file) {
            if ($file->getExtension() !== 'php') {
                continue; // Skip non-PHP files.
            }

            $realPath = $file->getRealPath();

            // Skip excluded directories.
            foreach ($excludedDirs as $excluded) {
                if (strpos($realPath, $excluded) === 0) {
                    continue 2;
                }
            }

            // Append file path and contents to compiled output.
            $relativePath = str_replace($basePath, '', $realPath);
            $compiledText .= "// File: {$relativePath}\n";
            $compiledText .= file_get_contents($realPath) . "\n\n";
        }

        // Save as TXT if enabled.
        if (!empty($outputFormats['txt'])) {
            $txtPath = $basePath . '/z-misc/compiled-code.txt';
            file_put_contents($txtPath, $compiledText);
        }

        // Save as PDF if enabled.
        if (!empty($outputFormats['pdf'])) {
            $pdfPath = $basePath . '/z-misc/compiled-code.pdf';

            // Wrap in <pre> to preserve formatting.
            $compiledHtml = "<pre style='white-space: pre-wrap; word-
wrap: break-word; overflow-wrap: break-word; margin: 0;'>" .
htmlspecialchars($compiledText) . "</pre>";

            // Generate PDF using Dompdf.
            $dompdf = new Dompdf();
            $dompdf->loadHtml($compiledHtml);
            $dompdf->setPaper('A4', 'portrait');
            $dompdf->render();

            file_put_contents($pdfPath, $dompdf->output());
        }
    }
}

// File: C:\laragon\www\multisite\wp-content\plugins\decentralized-user-
auth\decentralized-user-auth.php
<?php
/**
 * Plugin Name: Decentralized User Authentication
 * Description: Site-scoped user authentication for WordPress multisite.
 * Version: 1.0.0
 * Author: J. Lawrence Walkollie
 * Text Domain: dua
 * Domain Path: /languages
 * Network: true
 */

// Security: Prevent direct access to this file.
defined('ABSPATH') || exit;

// Plugin Constants
define('DUA_VERSION', '1.0.0');
define('DUA_PLUGIN_DIR', plugin_dir_path(__FILE__));
define('DUA_PLUGIN_URL', plugin_dir_url(__FILE__));

// Multisite Enforcement
if (!is_multisite()) {
    add_action('admin_notices', 'dua_show_multisite_required_notice');
    return;
}

/**
 * Displays an admin notice if WordPress Multisite is not enabled.
 * Hooked into 'admin_notices' during plugin bootstrap.
 *
 * @return void
 */
function dua_show_multisite_required_notice() {
    echo '<div class="notice notice-error"><p><strong>Decentralized User
Authentication</strong> requires WordPress Multisite to
function.</p></div>';
}

// Network Activation Check
add_action('admin_init', 'dua_check_network_activation');

/**
 * Displays a warning if the plugin is not network-activated.
 * Hooked into 'admin_init' to ensure plugin-wide availability.
 *
 * @return void
 */
function dua_check_network_activation() {
    if (!is_plugin_active_for_network(plugin_basename(__FILE__))) {
        add_action('admin_notices', function () {
            echo '<div class="notice notice-
warning"><p><strong>Decentralized User Authentication</strong> must be
network-activated to function properly across subsites.</p></div>';
        });
    }
}

// Activation and Deactivation Hooks
register_activation_hook(__FILE__, 'dua_activate');
register_deactivation_hook(__FILE__, 'dua_deactivate');

// Plugin Bootstrap
require_once DUA_PLUGIN_DIR . '/vendor/autoload.php';

use Dua\DuaPlugin;

// Instantiate the core plugin class.
new DuaPlugin();


// File: C:\laragon\www\multisite\wp-content\plugins\decentralized-user-
auth\inc\auth\DuaRoamingCookie.php
<?php
// Prevent direct access.
defined('ABSPATH') || exit;

/**
 * Manages roaming cookie authentication across subsites.
 * Handles session validation, login propagation, and secure cookie
signing.
 */
class DuaRoamingCookie {
    const COOKIE_NAME      = 'dua_roaming_user';
    const COOKIE_PATH      = '/';

    /**
     * Validates roaming session after WordPress cookie check.
     * Hooked into 'validate_auth_cookie'.
     *
     * @param int $userId
     * @return void
     */
    public static function onValidateSession($userId) {
        if (!self::isDomainCookieCompatible()) {
            return;
        }

        $cookie       = self::getValidCookie();
        $currentUser  = wp_get_current_user();
        $wpUserId     = $currentUser->ID;
        $isRoaming    = dua_is_roaming_user($currentUser);

        if ($cookie) {
            if (self::isLoggingOut()) {
                return;
            }

            if (!is_user_logged_in()) {
                self::setWpUser($cookie['user_id']);
            } elseif ($wpUserId !== $cookie['user_id']) {
                wp_logout();
                self::setWpUser($cookie['user_id']);
            }
        } elseif ($isRoaming && is_user_logged_in()) {
            wp_logout();
        }
    }

    /**
     * Sets roaming cookie after successful login.
     * Hooked into 'wp_login'.
     *
     * @param string   $userLogin
     * @param \WP_User $user
     * @return void
     */
    public static function onLogin($userLogin, $user) {
        if (dua_is_roaming_user($user)) {
            self::setCookie($user->ID);
        }
    }

    /**
     * Deletes roaming cookie before logout.
     * Hooked into 'wp_logout'.
     *
     * @return void
     */
    public static function onLogout() {
        self::deleteCookie();
    }

    /**
     * Checks if logout is in progress.
     * Prevents reauthentication during logout.
     *
     * @return bool
     */
    protected static function isLoggingOut() {
        return isset($_GET['action']) && $_GET['action'] === 'logout';
    }

    /**
     * Sets a signed roaming cookie for the user.
     *
     * @param int $userId
     * @return void
     */
    protected static function setCookie($user_id) {
        $iat     = time();
        $exp     = $iat + self::getLifetime();
        $nonce   = wp_generate_password(12, false);
        $payload = compact('user_id', 'iat', 'exp', 'nonce');

        $payload = apply_filters('dua_roaming_cookie_payload', $payload,
$user_id);
        $payload['sig'] = self::signPayload($payload);

        $cookieValue = rawurlencode(wp_json_encode($payload));

        setcookie(
            self::COOKIE_NAME,
            $cookieValue,
            [
                'expires'  => $exp,
                'path'     => self::COOKIE_PATH,
                'domain'   => self::getDomain(),
                'secure'   => true,
                'httponly' => true,
                'samesite' => 'Lax',
            ]
        );
    }

    /**
     * Validates roaming cookie and returns payload.
     *
     * @return array|false
     */
    protected static function getValidCookie() {
        if (empty($_COOKIE[self::COOKIE_NAME])) {
            return false;
        }

        $raw  = rawurldecode($_COOKIE[self::COOKIE_NAME]);
        $data = json_decode($raw, true);

        if (!is_array($data) || empty($data['sig'])) {
            return false;
        }

        $sig = $data['sig'];
        unset($data['sig']);

        $expectedSig = self::signPayload($data);

        if (!hash_equals($expectedSig, $sig)) {
            return false;
        }

        if (time() > $data['exp']) {
            return false;
        }

        return $data;
    }

    /**
     * Deletes the roaming cookie.
     *
     * @return void
     */
    public static function deleteCookie() {
        setcookie(
            self::COOKIE_NAME,
            '',
            [
                'expires'  => time() - 3600,
                'path'     => self::COOKIE_PATH,
                'domain'   => self::getDomain(),
                'secure'   => true,
                'httponly' => true,
                'samesite' => 'Lax',
            ]
        );
    }

    /**
     * Signs the cookie payload using HMAC.
     *
     * @param array $data
     * @return string
     */
    protected static function signPayload($data) {
        $json = wp_json_encode($data);
        return hash_hmac('sha256', $json, dua_get_roaming_secret_key());
    }

    /**
     * Returns cookie lifetime.
     *
     * @return int
     */
    protected static function getLifetime() {
        return dua_get_roaming_cookie_expiry();
    }

    /**
     * Returns wildcard domain for cookie scoping.
     *
     * @return string
     */
    protected static function getDomain() {
        $network = get_network();
        $domain  = $network->domain ?? $_SERVER['HTTP_HOST'] ??
'localhost';
        return '.' . ltrim(strtolower($domain), '.');
    }

    /**
     * Checks if domain matches host for cookie compatibility.
     *
     * @return bool
     */
    protected static function isDomainCookieCompatible() {
        $domain = self::getDomain();
        $host   = $_SERVER['HTTP_HOST'] ?? '';
        return stripos($host, ltrim($domain, '.')) !== false;
    }

    /**
```

```
     * Sets WordPress user context from cookie.
     *
     * @param int $userId
     * @return void
     */
    protected static function setWpUser($userId) {
        wp_set_auth_cookie($userId, true);
        wp_set_current_user($userId);
    }

    /**
     * Bypasses reauthentication if user is already validated.
     * Hooked into login flow.
     *
     * @return void
     */
    public static function maybeBypassReauth() {
        if (!is_user_logged_in()) {
            return;
        }

        if (isset($_GET['reauth']) && $_GET['reauth'] === '1' &&
isset($_GET['redirect_to'])) {
            wp_redirect(esc_url_raw($_GET['redirect_to']));
            exit;
        }
    }
}


// File: C:\laragon\www\multisite\wp-content\plugins\decentralized-user-
auth\inc\DuaActivator.php
<?php
// Prevent direct access.
defined('ABSPATH') || exit;

/**
 * Handles plugin activation and deactivation routines.
 * Creates required schema and installs MU plugin loader.
 */
class DuaActivator {

    /**
     * Runs on plugin activation.
     * Creates schema and installs MU plugin loader.
     *
     * @return void
     */
    public static function activate() {
        self::createUserSchema();
        self::injectSignupHook();
        self::installMuPlugin();
    }

    /**
     * Runs on plugin deactivation.
     * Removes MU plugin loader if present.
     *
     * @return void
     */
    public static function deactivate() {
        self::removeSignupHook();
        self::removeMuPlugin();
    }

    /**
     * Creates required columns in users and signups tables.
     * Adds site_id and main_id columns if missing.
     *
     * @return void
     */
    private static function createUserSchema() {
        global $wpdb;

        self::addColumnIfMissing($wpdb->prefix . 'users', 'site_id',
'BIGINT(20) UNSIGNED DEFAULT NULL');
        self::addColumnIfMissing($wpdb->prefix . 'users', 'main_id',
'BIGINT(20) UNSIGNED DEFAULT NULL');
        self::addColumnIfMissing($wpdb->prefix . 'usermeta', 'site_id',
'BIGINT(20) UNSIGNED DEFAULT NULL');
        self::addColumnIfMissing($wpdb->prefix . 'signups', 'site_id',
'BIGINT(20) UNSIGNED DEFAULT NULL');
    }

    /**
     * Adds a column to a table if it does not already exist.
     *
     * @param string $table
     * @param string $column
     * @param string $definition
     * @return void
     */
    private static function addColumnIfMissing($table, $column,
$definition) {
        global $wpdb;

        $exists = $wpdb->get_var(
            $wpdb->prepare(
                "SELECT COUNT(*) FROM INFORMATION_SCHEMA.COLUMNS
                 WHERE TABLE_NAME = %s AND COLUMN_NAME = %s AND
TABLE_SCHEMA = DATABASE()",
                $table,
                $column
            )
        );

        if ((int) $exists === 0) {
            $wpdb->query("ALTER TABLE {$table} ADD COLUMN {$column}
{$definition}");
        }
    }

    /**
     * Installs the MU plugin loader file.
     * Ensures the loader is placed in wp-content/mu-plugins.
     *
     * @return void
     */
    private static function installMuPlugin() {
        $muDir  = WP_CONTENT_DIR . '/mu-plugins';
        $source = DUA_PLUGIN_DIR . 'mu-loader/dua-mu-loader.php';
        $target = $muDir . '/dua-mu-loader.php';

        if (!file_exists($muDir)) {
            mkdir($muDir, 0755, true);
        }

        if (file_exists($target)) {
            unlink($target);
        }

        copy($source, $target);
    }

    /**
     * Remove the MU plugin loader file.
     * Ensures the loader is removed from wp-content/mu-plugins.
     *
     * @return void
     */
    private static function removeMuPlugin() {
        $muDir  = WP_CONTENT_DIR . '/mu-plugins';
        $target = $muDir . '/dua-mu-loader.php';

        if (file_exists($target)) {
            unlink($target);
        }
    }

    /**
     * Injects site-scoped signup validation hook into ms-functions.php.
     * Adds a single blank line above and below the injected block.
     * Preserves indentation of the original WordPress comment.
     */
    private static function injectSignupHook() {
        $path = ABSPATH . 'wp-includes/ms-functions.php';
        if (!file_exists($path)) return;

        $contents = file_get_contents($path);
        $anchor   = '// Has someone already signed up for this
username?';

        $injection =
            "// Injected by Decentralized User Authentication plugin:
site-scoped signup validation override\n" .
            "\t\$result =
apply_filters('dua_site_scoped_signup_validation', null, \$user_name,
\$user_email);\n" .
            "\tif (is_array(\$result)) {\n" .
            "\t\treturn apply_filters('wpmu_validate_user_signup',
\$result);\n" .
            "\t} // Decentralized User Authentication plugin ends
here.\n\n";

        if (
            strpos($contents, 'dua_site_scoped_signup_validation') ===
false &&
            strpos($contents, $anchor) !== false
        ) {
            $indentedAnchor = "\t" . $anchor;
            $contents = str_replace($anchor, $injection .
$indentedAnchor, $contents);
            file_put_contents($path, $contents);
        }
    }
    /**
     * Removes injected site-scoped signup validation hook from ms-
functions.php.
     * Matches the exact block including comment and spacing.
     */
    private static function removeSignupHook() {
        $path = ABSPATH . 'wp-includes/ms-functions.php';
        if (!file_exists($path)) return;

        $contents = file_get_contents($path);

        $pattern = '/\t\/\/ Injected by Decentralized User Authentication
plugin: site-scoped signup validation override\n' .
                   '\t\$result =
apply_filters\(\s*\'dua_site_scoped_signup_validation\',\s*null,\s*\$user
_name,\s*\$user_email\s*\);\n' .
                   '\tif\s*\(is_array\(\$result\)\)\s*\{\n' .
                   '\t\treturn
apply_filters\(\s*\'wpmu_validate_user_signup\',\s*\$result\s*\);\n' .
                   '\t\}\s*\/\/ Decentralized User Authentication plugin
ends here\.\n\n/';

        $contents = preg_replace($pattern, '', $contents);
        file_put_contents($path, $contents);
    }
}


// File: C:\laragon\www\multisite\wp-content\plugins\decentralized-user-
auth\inc\DuaAjaxController.php
<?php
namespace Dua;

use Dua\DuaUserLinker;

// Prevent direct access.
defined('ABSPATH') || exit;

/**
 * Handles AJAX endpoints for linking and unlinking user accounts.
 * Provides secure account linking across subsites.
 */
class DuaAjaxController {

    /**
     * Registers AJAX hooks for account linking operations.
     * Hooked during plugin initialization.
     *
     * @return void
     */
    public static function registerHooks() {
        $instance = new self();

        // Handles AJAX request to link a user account to a subsite.
        add_action('wp_ajax_dua_link_user_account', [$instance,
'linkUser']);

        // Handles AJAX request to unlink a user account from its main
account.
        add_action('wp_ajax_dua_unlink_user_account', [$instance,
'unlinkUser']);
```

```
        // Handles AJAX request to generate a remote login token URL.
        add_action('wp_ajax_dua_get_linked_account_token', [$instance,
'getToken']);

        // Intercepts login form for remote login flow (non-AJAX).
        add_action('login_form_remote_login', [DuaUserLinker::class,
'remoteLogin']);
    }

    /**
     * Links a user account to a subsite account.
     *
     * @return void
     */
    public function linkUser() {
        check_ajax_referer('dua_link_account', 'nonce');

        $mainUserId = absint($_POST['main_user_id'] ?? 0);
        $siteUrl    = esc_url_raw($_POST['site_url'] ?? '');
        $username   = sanitize_user($_POST['username'] ?? '');
        $password   = is_string($_POST['password'] ?? null) ?
$_POST['password'] : '';

        // Validate required input.
        if (!$mainUserId || !$username || !$siteUrl || !$password) {
            wp_send_json_error('Missing required data.');
        }

        // Ensure current user has permission to link.
        if ($mainUserId !== get_current_user_id() &&
!current_user_can('edit_user', $mainUserId)) {
            wp_send_json_error('Permission denied.');
        }

        $result = DuaUserLinker::linkAccount($mainUserId, $siteUrl,
$username, $password);

        is_wp_error($result)
            ? wp_send_json_error($result->get_error_message())
            : wp_send_json_success(['message' => 'Account linked
successfully.', 'account' => $result]);
    }

    /**
     * Unlinks a user account from its main account.
     *
     * @return void
     */
    public function unlinkUser() {
        $userId = absint($_POST['user_id'] ?? 0);

        // Validate user ID.
        if (!$userId) {
            wp_send_json_error('Invalid user ID.');
        }

        $result = DuaUserLinker::unlinkAccount($userId,
get_current_user_id());

        is_wp_error($result)
            ? wp_send_json_error($result->get_error_message())
            : wp_send_json_success('Account unlinked.');
    }

    /**
     * Generates a login token URL for remote authentication.
     *
     * @return void
     */
    public function getToken() {
        $userId = absint($_POST['user_id'] ?? 0);
        $siteId = absint($_POST['site_id'] ?? 0);

        // Validate input parameters.
        if (!$userId || !$siteId) {
            wp_send_json_error('Missing parameters.');
        }

        $url = DuaUserLinker::generateLoginUrl($userId, $siteId);

        wp_send_json_success(['login_url' => $url]);
    }
}


// File: C:\laragon\www\multisite\wp-content\plugins\decentralized-user-
auth\inc\DuaAuthToken.php
<?php
namespace Dua;

// Prevent direct access.
defined('ABSPATH') || exit;

/**
 * Handles generation and validation of authentication tokens.
 * Also provides lookup for linked subsite accounts.
 */
class DuaAuthToken {

    /**
     * Generates a signed authentication token for a user.
     * Includes user ID, site ID, IP address, and timestamp.
     *
     * @param int $userId
     * @param int $siteId
     * @return string Base64-encoded token
     */
    public static function generate($userId, $siteId) {
        $payload = [
            'user_id'   => $userId,
            'site_id'   => $siteId,
            'timestamp' => time(),
            'ip'        => $_SERVER['REMOTE_ADDR'],
        ];

        $secret = defined('DUA_AUTH_SECRET') ? DUA_AUTH_SECRET :
(wp_salt() . AUTH_KEY);
        $json   = json_encode($payload);
        $sig    = hash_hmac('sha256', $json, $secret);

        return base64_encode(json_encode([
            'data'      => $payload,
            'signature' => $sig,
        ]));
    }

    /**
     * Decodes and validates an authentication token.
     * Verifies signature and returns payload if valid.
     *
     * @param string $token
     * @return array|\WP_Error
     */
    public static function decode($token) {
        $raw = base64_decode($token, true);

        // Reject malformed base64.
        if (!$raw) {
            return new \WP_Error('invalid_token', 'Malformed token.');
        }

        $parsed = json_decode($raw, true);

        // Validate token structure.
        if (!is_array($parsed) || !isset($parsed['data'],
$parsed['signature'])) {
            return new \WP_Error('invalid_token', 'Invalid token
structure.');
        }

        $data     = $parsed['data'];
        $sig      = $parsed['signature'];
        $secret   = defined('DUA_AUTH_SECRET') ? DUA_AUTH_SECRET :
(wp_salt() . AUTH_KEY);
        $expected = hash_hmac('sha256', json_encode($data), $secret);

        // Verify signature.
        if (!hash_equals($expected, $sig)) {
            return new \WP_Error('invalid_token', 'Signature mismatch.');
        }

        return $data;
    }

    /**
     * Retrieves linked subsite accounts for a main user.
     * Cached for performance using wp_cache.
     *
     * @param int $mainUserId
     * @return array List of linked account objects
     */
    public static function getLinkedAccounts($mainUserId) {
        $cacheKey = 'dua_linked_' . $mainUserId;
        $accounts = wp_cache_get($cacheKey, 'dua');

        if (!$accounts) {
            global $wpdb;

            $accounts = $wpdb->get_results($wpdb->prepare("
                SELECT ID, user_login, user_email, site_id
                FROM {$wpdb->users}
                WHERE main_id = %d AND site_id != 1
            ", $mainUserId));

            $cacheExpiry = dua_get_cache_expiry();
            wp_cache_set($cacheKey, $accounts, 'dua', $cacheExpiry);
        }

        return $accounts;
    }
}


// File: C:\laragon\www\multisite\wp-content\plugins\decentralized-user-
auth\inc\DuaPlugin.php
<?php
namespace Dua;

// Prevent direct access.
defined('ABSPATH') || exit;

// Load required classes for plugin initialization.
use Dua\DuaUserHooks;
use Dua\DuaAjaxController;
use Dua\DuaSiteScopedSignup;
use Dua\Admin\DuaNetworkSettings;
use Dua\Admin\DuaAddUserPageController;
use Dua\Admin\DuaLinkedAccountsProfile;

/**
 * Core plugin bootstrap class.
 *
 * Handles dependency loading, localization, asset registration,
 * and component initialization.
 */
class DuaPlugin {

    /**
     * Initializes the plugin lifecycle.
     * Called during plugin instantiation from the main file.
     */
    public function __construct() {
        $this->loadDependencies();
        $this->registerTextDomain();
        $this->enqueueAdminAssets();
        $this->registerHooks();
        $this->initializeComponents();
    }

    /**
     * Loads internal utility files and overrides.
     * Includes helper functions and pluggable overrides.
     */
    private function loadDependencies() {
        require_once DUA_PLUGIN_DIR . 'inc/utils.php';
        require_once DUA_PLUGIN_DIR . 'overrides/pluggable.php';
    }

    /**
     * Registers plugin text domain for localization.
     * Hooked into 'plugins_loaded'.
     */
    private function registerTextDomain() {
        add_action('plugins_loaded', [$this, 'loadTextDomain']);
    }

    /**
     * Loads plugin translation files.
     *
     * @return void
     */
    public function loadTextDomain() {
        load_plugin_textdomain('dua', false,
dirname(plugin_basename(__FILE__)) . '/../languages');
    }

    /**
     * Registers admin asset loading hook.
     * Hooked into 'admin_enqueue_scripts'.
     */
    private function enqueueAdminAssets() {
        add_action('admin_enqueue_scripts', [$this, 'loadAdminAssets']);
    }

    /**
     * Loads admin-specific JavaScript and CSS assets.
     * Only enqueued specific site and network admin page.
     *
     * @param string $hook Current admin page hook.
     * @return void
     */
    public function loadAdminAssets($hook) {
        // Pages in single-site and network admin that need the assets.
        $singleSiteHooks = ['profile.php', 'user-edit.php'];
        $networkHooks = ['settings.php', 'site-info.php', 'site-
users.php'];

        // Determine if we're on a supported single-site or network admin
page.
        $loadAssetsForSite = in_array($hook, $singleSiteHooks, true);
        $loadAssetsForNetwork = is_network_admin() && in_array($hook,
$networkHooks, true);

        // Load assets only if the current page matches one of the
allowed contexts.
        if ($loadAssetsForSite || $loadAssetsForNetwork) {
            wp_enqueue_script(
                'dua-admin-js',
                DUA_PLUGIN_URL . '/assets/js/admin.js',
                ['jquery'],
                null,
                true
            );

            wp_enqueue_style(
                'dua-admin-css',
                DUA_PLUGIN_URL . '/assets/css/admin.css',
                [],
                null
            );
        }
    }

    /**
     * Registers global plugin hooks.
     *
     * Includes core patching logic and UI suppression for multisite
environments.
     * Called during plugin bootstrap to ensure early execution.
     *
     * @return void
     */
    public static function registerHooks() {
        // Re-inject signup hook after WordPress core upgrade.
        add_action('upgrader_process_complete',
'dua_after_wordpress_upgrade', 10, 2);
    }

    /**
     * Initializes plugin components and registers hooks.
     * Instantiates admin pages and registers core hooks.
     */
    private function initializeComponents() {
        DuaUserHooks::registerHooks();
        DuaAjaxController::registerHooks();
        DuaSiteScopedSignup::registerHooks();

        new DuaNetworkSettings();
        new DuaLinkedAccountsProfile();
        new DuaAddUserPageController();
    }
}


// File: C:\laragon\www\multisite\wp-content\plugins\decentralized-user-
auth\inc\DuaSiteScopedSignup.php
<?php
namespace Dua;

// Prevent direct access.
defined('ABSPATH') || exit;

/**
 * Handles site-scoped validation and metadata storage during user
signup.
 * Injects custom validation logic via dua_site_scoped_signup_validation
filter.
 */
class DuaSiteScopedSignup {

    /**
     * Registers WordPress hooks for multisite signup behavior.
     * Hooked during plugin initialization.
     *
     * @return void
     */
    public static function registerHooks() {
        // Injects full site-scoped validation logic.
        add_filter('dua_site_scoped_signup_validation', [self::class,
'validateSignup'], 10, 3);

        // Stores site_id in wp_signups table after a user signs up.
        add_action('after_signup_user', [self::class, 'storeSiteId'], 10,
4);
    }

    /**
     * Performs full site-scoped validation for username and email.
     * Replaces default signup reservation logic.
     *
     * @param mixed  $override Null or array to override validation.
     * @param string $user_name
     * @param string $user_email
     * @return array|null
     */
    public static function validateSignup($override, $user_name,
$user_email) {
        global $wpdb;

        $errors         = new \WP_Error();
        $orig_username  = $user_name;
        $site_id        = get_current_blog_id();

        // Check username reservation scoped to site
        $signup = $wpdb->get_row(
            $wpdb->prepare(
                "SELECT * FROM {$wpdb->signups} WHERE user_login = %s AND
site_id = %d",
                $user_name,
                $site_id
            )
        );

        if ($signup instanceof \stdClass) {
            $registered_at = mysql2date('U', $signup->registered);
            $now           = time();
            $diff          = $now - $registered_at;

            if ($diff > 2 * DAY_IN_SECONDS) {
                $wpdb->delete($wpdb->signups, ['user_login' =>
$user_name, 'site_id' => $site_id]);
            } else {
                $errors->add('user_name', __('That username is currently
reserved but may be available in a couple of days.'));
            }
        }

        // Check email reservation scoped to site
        $signup = $wpdb->get_row(
            $wpdb->prepare(
                "SELECT * FROM {$wpdb->signups} WHERE user_email = %s AND
site_id = %d",
                $user_email,
                $site_id
            )
        );

        if ($signup instanceof \stdClass) {
            $diff = time() - mysql2date('U', $signup->registered);

            if ($diff > 2 * DAY_IN_SECONDS) {
                $wpdb->delete($wpdb->signups, ['user_email' =>
$user_email, 'site_id' => $site_id]);
            } else {
                $errors->add('user_email', __('That email address has
already been used. Please check your inbox for an activation email. It
will become available in a couple of days if you do nothing.'));
            }
        }

        return [
            'user_name'     => $user_name,
            'orig_username' => $orig_username,
            'user_email'    => $user_email,
            'errors'        => $errors,
        ];
    }

    /**
     * Stores site_id in wp_signups after signup creation.
     *
     * @param string $userLogin
     * @param string $userEmail
     * @param string $activationKey
     * @param array  $meta Optional metadata (unused).
     * @return void
     */
    public static function storeSiteId($userLogin, $userEmail,
$activationKey, $meta = []) {
        global $wpdb;

        $siteId = get_current_blog_id();

        $wpdb->update(
            $wpdb->signups,
            ['site_id' => $siteId],
            [
                'user_login'     => $userLogin,
                'activation_key' => $activationKey,
            ]
        );
    }
}


// File: C:\laragon\www\multisite\wp-content\plugins\decentralized-user-
auth\inc\DuaUserHooks.php
<?php
namespace Dua;

// Prevent direct access.
defined('ABSPATH') || exit;

/**
 * Registers core user-related hooks for site-scoped behavior.
 * Handles user registration, deletion, and blog removal logic.
 */
class DuaUserHooks {

    /**
     * Registers WordPress hooks for user lifecycle events.
     * Hooked during plugin initialization.
     *
     * @return void
     */
    public static function registerHooks() {
        // Assigns site_id to newly registered users immediately after
registration.
        add_action('user_register', [self::class, 'assignSiteId']);

        // Blocks deletion of users not scoped to the current site.
        // Ensures site isolation during user lifecycle events.
        add_action('delete_user', [self::class,
'preventCrossSiteDeletion']);

        // Prevents unauthorized removal of users from blogs unless
```

```
scoped.
        // Runs before WordPress removes a user from a site.
        add_filter('pre_remove_user_from_blog', [self::class,
'preventUnauthorizedRemoval'], 10, 2);

        // Registers cleanup flag and shutdown logic when a user is
removed from a blog.
        add_action('remove_user_from_blog', [self::class,
'markUserForCleanup'], 10, 3);

        // Runs early on every request to ensure orphaned users flagged
are cleaned up.
        // This acts as a fallback in case shutdown-based cleanup was
skipped.
        add_action('muplugins_loaded', [self::class,
'checkOrphanedUsersForCurrentSite']);
    }

    /**
     * Assigns the current site ID to newly registered users.
     *
     * @param int $userId ID of the newly registered user.
     * @return void
     */
    public static function assignSiteId($userId) {
        global $wpdb;
        $siteId = get_current_blog_id();

        // Update the site_id column for the new user.
        $wpdb->update(
            $wpdb->users,
            ['site_id' => $siteId],
            ['ID' => $userId],
            ['%d'],
            ['%d']
        );
    }

    /**
     * Prevents deletion of users who are not scoped to the current site.
     *
     * @param int $userId ID of the user being deleted.
     * @return void
     */
    public static function preventCrossSiteDeletion($userId) {
        global $wpdb;
        $siteId = get_current_blog_id();

        // Retrieve site_id and main_id for the user.
        $userData = $wpdb->get_row($wpdb->prepare("
            SELECT site_id, main_id FROM {$wpdb->users}
            WHERE ID = %d
        ", $userId));

        if (!$userData) {
            return;
        }

        // Block deletion if user is not scoped to current site.
        if ((int) $userData->site_id !== $siteId) {
            wp_die(__('Cannot delete user outside your site scope.',
'decentralized-user-auth'));
```

```
        }
    }

    /**
     * Prevents unauthorized removal of users from a blog.
     *
     * @param int $userId ID of the user being removed.
     * @param int $blogId ID of the target blog.
     * @return WP_Error|null
     */
    public static function preventUnauthorizedRemoval($userId, $blogId) {
        // Allow removal from network admin.
        if (is_network_admin()) {
            return null;
        }

        global $wpdb;

        // Retrieve site_id for the user.
        $userData = $wpdb->get_row($wpdb->prepare("
            SELECT site_id FROM {$wpdb->users}
            WHERE ID = %d
        ", $userId));

        if (!isset($userData->site_id)) {
            return null;
        }

        // Block removal if user is not scoped to target blog.
        if ((int) $userData->site_id !== (int) $blogId) {
            return new \WP_Error(
                'unauthorized_removal',
                __('Unauthorized removal: User is not scoped to this
site.', 'decentralized-user-auth')
            );
        }

        return null;
    }

    /**
     * Marks a user for deferred cleanup after being removed from a blog.
     * Writes a temporary flag to the dua-data folder and registers
shutdown logic.
     *
     * @param int $userId ID of the user being removed.
     * @param int $blogId ID of the blog the user is being removed from.
     * @param int $reassign ID of the user to reassign content to (if
any).
     * @return void
     */
    public static function markRemovedBlogUserForCleanup($userId,
$blogId, $reassign) {
        $path = WP_CONTENT_DIR . '/dua-data/pending-user-cleanup.php';

        // Ensure folder exists.
        if (!is_dir(dirname($path))) {
            mkdir(dirname($path), 0755, true);
        }

        // Ensure file exists.
        if (!file_exists($path)) {
            file_put_contents($path, "<?php\nreturn [];\n");
        }

        // Load current flags.
        $flags = include $path;

        // Add user to site’s cleanup list.
        $flags[$blogId] = isset($flags[$blogId])
            ? array_unique(array_merge($flags[$blogId], [$userId]))
            : [$userId];

        // Write updated flags back to file.
        file_put_contents($path, "<?php\nreturn " . var_export($flags,
true) . ";\n");

        // Register shutdown logic for this user.
        add_action('shutdown', function () use ($userId, $blogId, $path)
{
            self::cleanupRemovedBlogUserIfOrphaned($userId, $blogId,
$path);
        });
    }

    /**
     * Deletes a user from the network if they no longer belong to any
site.
     * Cleans up the temporary flag after successful deletion.
     *
     * @param int    $userId ID of the user to check and possibly delete.
     * @param int    $blogId ID of the blog the user was removed from.
     * @param string $path   Full path to the cleanup flag file.
     * @return void
     */
    public static function cleanupRemovedBlogUserIfOrphaned($userId,
$blogId, $path) {
        $blogs = get_blogs_of_user($userId);

        if (empty($blogs)) {
            wpmu_delete_user($userId);
        }

        // Reload flags to ensure latest state.
        $flags = file_exists($path) ? include $path : [];

        if (isset($flags[$blogId])) {
            // Remove user from the cleanup list.
            $flags[$blogId] = array_filter($flags[$blogId], fn($id) =>
$id !== $userId);

            // Remove empty site entry.
            if (empty($flags[$blogId])) {
                unset($flags[$blogId]);
            }

            // Write updated flags back to file.
            file_put_contents($path, "<?php\nreturn " .
var_export($flags, true) . ";\n");
        }
    }

    /**
     * Checks for orphaned users flagged for the current site and deletes
```

```
them if necessary.
     * Runs early on every request to ensure cleanup even if shutdown was
skipped.
     *
     * @return void
     */
    public static function checkOrphanedUsersForCurrentSite() {
        $siteId = get_current_blog_id();
        $path = WP_CONTENT_DIR . '/dua-data/pending-user-cleanup.php';

        if (!file_exists($path)) {
            return;
        }

        $flags = include $path;

        if (!isset($flags[$siteId]) || empty($flags[$siteId])) {
            return;
        }

        $changed = false;

        // Check if flagged users for current site are orphaned and
delete them.
        foreach ($flags[$siteId] as $index => $userId) {
            $blogs = get_blogs_of_user($userId);

            if (empty($blogs)) {
                wpmu_delete_user($userId);
                unset($flags[$siteId][$index]);
                $changed = true;
            }
        }

        // Clean up empty site entry
        if (empty($flags[$siteId])) {
            unset($flags[$siteId]);
            $changed = true;
        }

        // Write updated flags back to file
        if ($changed) {
            file_put_contents($path, "<?php\nreturn " .
var_export($flags, true) . ";\n");
        }
    }
}


// File: C:\laragon\www\multisite\wp-content\plugins\decentralized-user-
auth\inc\DuaUserLinker.php
<?php
namespace Dua;

use Dua\DuaAuthToken;
use WP_Error;

// Prevent direct access.
defined('ABSPATH') || exit;

/**
 * Handles linking and unlinking of user accounts across subsites.
 * Also manages remote login via token-based authentication.
 */
class DuaUserLinker {

    /**
     * Links a subsite account to a main user account.
     * Validates credentials and updates user metadata.
     *
     * @param int    $mainUserId
     * @param string $siteUrl
     * @param string $username
     * @param string $password
     * @return array|\WP_Error
     */
    public static function linkAccount($mainUserId, $siteUrl, $username,
$password) {
        $parsedHost = parse_url($siteUrl, PHP_URL_HOST);
        $site       = get_site_by_path($parsedHost, '/');

        // Validate site existence.
        if (!$site) {
            return new WP_Error('invalid_site', 'Invalid subsite URL.');
        }

        $blogId = (int) $site->blog_id;

        // Attempt to retrieve user by login or email.
        $user = get_user_by('login', $username, $blogId) ?:
get_user_by('email', $username, $blogId);

        // Validate credentials.
        if (!$user || !wp_check_password($password, $user->user_pass,
$user->ID)) {
            return new WP_Error('auth_failed', 'Authentication failed.');
        }

        // Check for existing linkage.
        $existingMainId = (int) $user->main_id;
        if ($existingMainId && $existingMainId !== $mainUserId) {
            return new WP_Error('already_linked', 'This account is
already linked to another user.');
        }

        global $wpdb;

        // Link account by updating main_id.
        $wpdb->update($wpdb->users, ['main_id' => $mainUserId], ['ID' =>
$user->ID]);

        return [
            'site_id'    => $blogId,
            'site_url'   => $siteUrl,
            'user_login' => $user->user_login,
            'user_email' => $user->user_email,
            'ID'         => $user->ID
        ];
    }

    /**
     * Unlinks a subsite account from its main user.
     * Requires ownership or network-level capability.
     *
     * @param int $userId
     * @param int $currentUserId
     * @return bool|\WP_Error
     */
    public static function unlinkAccount($userId, $currentUserId) {
        global $wpdb;

        $linkedMainId = (int) $wpdb->get_var($wpdb->prepare(
            "SELECT main_id FROM {$wpdb->users} WHERE ID = %d", $userId
        ));

        $isMainUser      = ($linkedMainId === $currentUserId);
        $hasNetworkCaps  = is_main_site() &&
current_user_can('manage_network_users');

        // Validate permission to unlink.
        if (!$isMainUser && !$hasNetworkCaps) {
            return new WP_Error('unauthorized', 'Unauthorized unlink
attempt.');
        }

        // Remove linkage.
        $wpdb->update($wpdb->users, ['main_id' => null], ['ID' =>
$userId]);

        return true;
    }

    /**
     * Generates a remote login URL for a subsite account.
     *
     * @param int $userId
     * @param int $siteId
     * @return string
     */
    public static function generateLoginUrl($userId, $siteId) {
        $token = DuaAuthToken::generate($userId, $siteId);
        return get_site_url($siteId) . '/wp-
login.php?action=remote_login&token=' . urlencode($token);
    }

    /**
     * Handles remote login via token authentication.
     * Validates token, IP, and rate limits before logging in.
     *
     * @return void
     */
    public static function remoteLogin() {
        $token = $_GET['token'] ?? '';
        $data  = DuaAuthToken::decode($token);

        // Validate token structure.
        if (is_wp_error($data)) {
            wp_redirect(home_url('/?error=invalid_token'));
            exit;
        }

        $userId   = (int) $data['user_id'];
        $siteId   = (int) $data['site_id'];
        $timestamp= (int) $data['timestamp'];
        $ip       = $data['ip'];
        $expiry   = dua_get_remote_login_token_expiry();
        // Validate IP match.
        if ($_SERVER['REMOTE_ADDR'] !== $ip) {
            wp_redirect(home_url('/?error=ip_mismatch'));
            exit;
        }

        // Validate token expiry.
        if (time() - $timestamp > $expiry) {
            wp_redirect(home_url('/?error=token_expired'));
            exit;
        }

        // Rate limiting
        $limitKey = "dua_login_attempts_{$userId}";
        $attempts = (int) get_transient($limitKey);

        if ($attempts >= 5) {
            wp_redirect(home_url('/?error=rate_limited'));
            exit;
        }

        $cacheExpiry = dua_get_cache_expiry();
        set_transient($limitKey, $attempts + 1, $cacheExpiry);

        // Authenticate user.
        $user = get_user_by('ID', $userId);
        if ($user) {
            wp_set_auth_cookie($userId, true);
            wp_redirect(home_url());
            exit;
        }

        wp_redirect(home_url('/?error=user_not_found'));
        exit;
    }
}


// File: C:\laragon\www\multisite\wp-content\plugins\decentralized-user-
auth\inc\utils.php
<?php
/**
 * Utility functions for Decentralized User Authentication.
 * Includes caching helpers and roaming user detection.
 */

// Prevent direct access.
defined('ABSPATH') || exit;

/**
 * Determines whether a user is considered roaming.
 *
 * A roaming user is typically a super admin across the network.
 * Developers may override this logic via the 'dua_is_roaming_user'
filter.
 *
 * @param object $user WP_User or stdClass object.
 * @return bool
 */
function dua_is_roaming_user($user) {
    $is_roaming = false;
    // Check if user is a super admin.
    if (is_object($user) && !empty($user->user_login)) {
        $super_admins = get_super_admins();
        $is_roaming   = in_array($user->user_login, $super_admins, true);
    }

    return apply_filters('dua_is_roaming_user', $is_roaming, $user);
}

/**
 * Retrieves the cached expiry time for plugin data.
 *
 * Falls back to site option if transient is missing.
 *
 * @return int Expiry time in seconds.
 */
function dua_get_cache_expiry() {
    $cached = get_transient('dua_cache_expiry_cached');

    // Return cached value if available.
    if ($cached !== false) {
        return $cached;
    }

    // Fallback to site option and cache it.
    $expiry = get_site_option('dua_cache_expiry', 3600);
    set_transient('dua_cache_expiry_cached', $expiry, HOUR_IN_SECONDS);

    return $expiry;
}

/**
 * Retrieves a site option with transient caching.
 *
 * @param string $key           Option key.
 * @param mixed  $default       Default value if option is missing.
 * @param string $transient_key Transient cache key.
 * @return mixed
 */
function dua_get_cached_option($key, $default, $transient_key) {
    $cached = get_transient($transient_key);

    // Return cached value if available.
    if ($cached !== false) {
        return $cached;
    }

    // Fallback to site option and cache it.
    $value = get_site_option($key, $default);
    set_transient($transient_key, $value, dua_get_cache_expiry());

    return $value;
}

/**
 * Retrieves the roaming secret key from network settings.
 * Falls back to default if not set. Cached via transient.
 *
 * @return string
 */
function dua_get_roaming_secret_key() {
```

```php
    $default = 'dua-super-consistent-network-secret';
    return dua_get_cached_option('dua_roaming_secret_key', $default,
'dua_roaming_secret_key_cached');
}

/**
 * Retrieves the roaming cookie duration.
 * Cached via 'dua_roaming_cookie_expiry_cached'.
 *
 * @return int
 */
function dua_get_roaming_cookie_expiry() {
    return dua_get_cached_option('dua_roaming_cookie_expiry', 60,
'dua_roaming_cookie_expiry_cached');
}

/**
 * Retrieves the remote login token expiry duration.
 * Cached via 'dua_remote_login_token_expiry_cached'.
 *
 * @return int
 */
function dua_get_remote_login_token_expiry() {
    return dua_get_cached_option('dua_remote_login_token_expiry', 60,
'dua_remote_login_token_expiry_cached');
}

/**
 * Retrieves the maximum allowed login attempts.
 * Cached via 'dua_rate_limit_max_cached'.
 *
 * @return int
 */
function dua_get_rate_limit_max() {
    return dua_get_cached_option('dua_rate_limit_max', 5,
'dua_rate_limit_max_cached');
}

/**
 * Retrieves the wait time after rate limit is triggered.
 * Cached via 'dua_rate_limit_wait_cached'.
 *
 * @return int
 */
function dua_get_rate_limit_wait() {
    return dua_get_cached_option('dua_rate_limit_wait', 300,
'dua_rate_limit_wait_cached');
}

/**
 * Handles plugin activation logic.
 *
 * Loads the DuaActivator class manually and triggers schema setup
 * and MU plugin installation. This class is not autoloaded to avoid
 * unnecessary runtime overhead.
 *
 * @return void
 */
function dua_activate() {
    require_once DUA_PLUGIN_DIR . '/inc/DuaActivator.php';
    DuaActivator::activate();
}
```

```
/**
 * Handles plugin deactivation logic.
 *
 * Loads the DuaActivator class manually and removes the MU plugin
loader.
 * This ensures clean teardown without autoloading unused classes.
 *
 * @return void
 */
function dua_deactivate() {
    require_once DUA_PLUGIN_DIR . '/inc/DuaActivator.php';
    DuaActivator::deactivate();
}

/**
 * Re-injects site-scoped signup validation hook after WordPress core
upgrade.
 *
 * WordPress overwrites core files during upgrades, including ms-
functions.php.
 * This function ensures the Decentralized User Authentication patch is
reapplied
 * immediately after a core update completes.
 *
 * @param WP_Upgrader $upgrader WP_Upgrader instance.
 * @param array       $options  Upgrade context including 'action' and
'type'.
 * @return void
 */
function dua_after_wordpress_upgrade($upgrader, $options) {
    if (
        isset($options['action'], $options['type']) &&
        $options['action'] === 'update' &&
        $options['type'] === 'core'
    ) {
        require_once DUA_PLUGIN_DIR . '/inc/DuaActivator.php';
        DuaActivator::injectSignupHook();
    }
}


// File: C:\laragon\www\multisite\wp-content\plugins\decentralized-user-
auth\mu-loader\dua-mu-loader.php
<?php
/**
 * MU plugin: Early cookie scoping for multisite isolation.
 *
 * Sets site-specific cookie constants before WordPress loads fully.
 * Enables roaming support via externally validated cookies and filters.
 * Requires a shared network salt for consistent token signing.
 */

// Prevent direct access.
defined('ABSPATH') || exit;

/**
 * Step 1: Define default cookie constants early.
 *
 * These constants are scoped per site and per browser instance.
 * They may be overridden later if roaming logic applies.
 */
```

```
$host       = $_SERVER['HTTP_HOST'] ?? 'default.local';
$cleanHost  = preg_replace('/[^a-z0-9.-]/', '', strtolower($host));
$blogId     = get_current_blog_id();
$userAgent  = $_SERVER['HTTP_USER_AGENT'] ?? '';
$uaHash     = substr(md5($userAgent), 0, 8); // Short hash per browser

// Construct cookie suffix using host, site ID, and UA hash.
$cookieSuffix = md5($cleanHost) . "_site_{$blogId}_{$uaHash}";
$cookieDomain = $cleanHost;

// Define WordPress cookie constants for this site context.
define('COOKIEHASH', $cookieSuffix);
define('LOGGED_IN_COOKIE', 'wordpress_logged_in_' . COOKIEHASH);
define('AUTH_COOKIE', 'wordpress_' . COOKIEHASH);
define('SECURE_AUTH_COOKIE', 'wordpress_sec_' . COOKIEHASH);
define('USER_COOKIE', 'wordpress_user_' . COOKIEHASH);
define('PASS_COOKIE', 'wordpress_pass_' . COOKIEHASH);
define('TEST_COOKIE', 'wordpress_test_cookie_' . COOKIEHASH);
define('COOKIE_DOMAIN', $cookieDomain);
define('COOKIEPATH', '/');
define('SITECOOKIEPATH', '/');

/**
 * Step 2: Include the roaming cookie handler, and necessary files.
 * This class manages cross-site authentication via signed cookies.
 */
require_once WP_PLUGIN_DIR . '/decentralized-user-auth/inc/utils.php';
require_once WP_PLUGIN_DIR . '/decentralized-user-
auth/inc/auth/DuaRoamingCookie.php';

/**
 * Step 3: Hook into WordPress login and session lifecycle.
 * These hooks enable roaming user detection and cookie propagation.
 */
// Fires after successful login and cookie creation.
// Used to set roaming cookie for cross-site authentication.
add_action('wp_login', ['DuaRoamingCookie', 'onLogin'], 10, 2);

// Fires when WordPress sets the current user.
// Used to validate roaming cookie and switch user context if needed.
add_action('set_current_user', ['DuaRoamingCookie',
'onValidateSession']);

// Fires during login form initialization.
// Used to bypass reauthentication if roaming session is already valid.
add_action('login_init', ['DuaRoamingCookie', 'maybeBypassReauth']);

// Fires before logout.
// Used to delete roaming cookie and clean up session state.
add_action('wp_logout', ['DuaRoamingCookie', 'onLogout']);


// File: C:\laragon\www\multisite\wp-content\plugins\decentralized-user-
auth\overrides\Dua_WP_User.php
<?php
namespace Dua\Overrides;

// Exit if accessed directly
defined('ABSPATH') || exit;

class Dua_WP_User {
    public static function get_data_by( $field, $value, $site_id = null )
```

```
{
        global $wpdb;

        if ( 'ID' === $field ) {
            $field = 'id';
        }

        if ( 'id' === $field ) {
            if ( ! is_numeric( $value ) || (int) $value < 1 ) {
                return false;
            }
            $value = (int) $value;
        } else {
            $value = trim( $value );
        }

        if ( ! $value || is_null( $site_id ) ) {
            return false;
        }

        switch ( $field ) {
            case 'id':
                $db_field = 'ID';
                break;
            case 'slug':
                $value    = sanitize_title( $value );
                $db_field = 'user_nicename';
                break;
            case 'email':
                $value    = sanitize_email( $value );
                $db_field = 'user_email';
                break;
            case 'login':
                $value    = sanitize_user( $value );
                $db_field = 'user_login';
                break;
            default:
                return false;
        }

        $cache_key = "dua_user_{$field}_{$value}_site_{$site_id}";
        $user = wp_cache_get($cache_key, 'dua');

        if (!$user) {
            $user = $wpdb->get_row(
                $wpdb->prepare(
                    "SELECT * FROM {$wpdb->users} WHERE $db_field = %s 
AND site_id = %d LIMIT 1",
                    $value,
                    $site_id
                )
            );

            // Fallback to site_id = 1 if user not found
            if ( ! $user && $site_id !== 1 ) {
                $main_site_id = 1;
                $fallback_user = $wpdb->get_row(
                    $wpdb->prepare(
                        "SELECT * FROM {$wpdb->users} WHERE $db_field = 
%s AND site_id = %d LIMIT 1",
                        $value,
                        $main_site_id
                    )
                );

                /**
                 * Only allow fallback if user is roaming user.
                 */
                if ( $fallback_user && dua_is_roaming_user(
$fallback_user ) ) {
                    $user = $fallback_user;
                }
            }

            if ($user) {
                $cache_expiry = dua_get_cache_expiry();
                wp_cache_set($cache_key, $user, 'dua', $cache_expiry);
            }
        }

        if ( ! $user ) {
            return false;
        }

        update_user_caches( $user );

        return $user;
    }
}


// File: C:\laragon\www\multisite\wp-content\plugins\decentralized-user-
auth\overrides\pluggable.php
<?php
/**
 * These are default wordpress functions being replaced by this plugin.
 *
 * @package WordPress
 */

// Load require classes.
use Dua\Overrides\Dua_WP_User;

if ( ! function_exists( 'get_user_by' ) ) :
    /**
     * Retrieves user info by a given field, scoped by site ID.
     *
     * @param string     $field   Field to search by: id | ID | slug |
email | login.
     * @param int|string $value   Field value.
     * @param int|null   $site_id Optional site ID. Defaults to current
site.
     * @return WP_User|false
     */
    function get_user_by( $field, $value, $site_id = null ) {
        if ( $site_id === null ) {
            $site_id = get_current_blog_id();
        }

        $userdata = Dua_WP_User::get_data_by( $field, $value, $site_id );

        if ( ! $userdata ) {
            return false;
        }
        //$user = new WP_User();
        $user = new WP_User($userdata->ID);
        $user->init( $userdata );

        return $user;
    }
endif;


// File: C:\laragon\www\multisite\wp-content\plugins\decentralized-user-
auth\uninstall.php
```