source: suretriggers / trunk / src / Controllers / RestController.php

Last change on this file was 3270907, checked in by brainstormworg, 32 hours ago

Update to version 1.0.81 from GitHub

File size: 15.3 KB

```
<?php
/**
 * RestController.
 * php version 5.6
 *
 * @category RestController
 * @package  SureTriggers
 * @author   BSF <username@example.com>
 * @license  https://www.gnu.org/licenses/gpl-3.0.html GPLv3
 * @link     https://www.brainstormforce.com/
 * @since    1.0.0
 */

namespace SureTriggers\Controllers;

use Exception;
use SureTriggers\Integrations\WordPress\WordPress;
use SureTriggers\Traits\SingletonLoader;
use SureTriggers\Models\SaasApiToken;
use WP_REST_Request;
use WP_REST_Response;
use WP_Error;

/**
 * RestController
 *
 * @category RestController
 * @package  SureTriggers
 * @author   BSF <username@example.com>
 * @license  https://www.gnu.org/licenses/gpl-3.0.html GPLv3
 * @link     https://www.brainstormforce.com/
 * @since    1.0.0
 */
class RestController {

        /**
         * Access token for authentication.
         *
         * @var string $acccess_token
         */
        private $secret_key;

        use SingletonLoader;

        /**
         * Initialise data.
         */
        public function __construct() {
                $this->secret_key = SaasApiToken::get();
```

```
50                 add_filter( 'determine_current_user', [ $this, 'basic_auth_handler' ], 20 );
51                 add_filter( 'debug_information', [ $this, 'sure_triggers_connection_info' ] );
52         }
53
54         /**
55          * Permission callback for rest api after determination of current user.
56          *
57          * @param WP_REST_Request $request Request.
58          */
59         public function autheticate_user( $request ) {
60                 $secret_key = $request->get_header( 'st_authorization' );
61
62                 if ( ! is_string( $secret_key ) || empty( $secret_key ) || empty( $this->secret_key ) ) {
63                         return false;
64                 }
65
66                 list($secret_key) = sscanf( $secret_key, 'Bearer %s' );
67
68                 if ( empty( $secret_key ) ) {
69                         return false;
70                 }
71
72                 if ( $this->secret_key !== $secret_key ) {
73                         return false;
74                 }
75
76                 return hash_equals( $this->secret_key, $secret_key );
77         }
78
79         /**
80          * Create WP Connection.
81          *
82          * @param WP_REST_Request $request Request data.
83          * @return WP_REST_Response
84          */
85         public function create_wp_connection( $request ) {
86
87                 $user_agent = $request->get_header( 'user-agent' );
88                 if ( 'OttoKit' !== $user_agent && 'SureTriggers' !== $user_agent ) {
89                         return new WP_REST_Response(
90                                 [
91                                         'success' => false,
92                                         'data'    => 'Unauthorized',
93                                 ],
94                                 403
95                         );
96                 }
97                 $params = $request->get_json_params();
98
99                 $username = isset( $params['wp-username'] ) ? sanitize_text_field( $params['wp-
username'] ) : '';
100                $password = isset( $params['wp-password'] ) ? $params['wp-password'] : '';
101
102                if ( empty( $username ) || empty( $password ) ) {
103                        return new WP_REST_Response(
104                                [
105                                        'success' => false,
106                                        'data'    => 'Username and password are required.',
107                                ],
108                                400
109                        );
110                }
111
```

```
Line
112                 $user = wp_authenticate_application_password( null, $username, $password );
113
114                 if ( is_wp_error( $user ) ) {
115                         return new WP_REST_Response(
116                                 [
117                                         'success' => false,
118                                         'data'    => 'Invalid username or password.',
119                                 ],
120                                 403
121                         );
122                 }
123
124                 $connection_status = $request->get_param( 'connection-status' );
125                 $access_key        = $request->get_param( 'sure-triggers-access-key' );
126                 $connected_email   = $request->get_param( 'connected_email' );
127
128                 if ( false === $connection_status ) {
129                         $access_key = 'connection-denied';
130                 }
131
132 $connected_email_id = isset( $connected_email ) ? sanitize_email( wp_unslash( $connected_email ) ) : '';
133
134                 if ( isset( $access_key ) ) {
135                         SaasApiToken::save( $access_key );
136                 }
137                 OptionController::set_option( 'connected_email_key', $connected_email_id );
138
139                 return new WP_REST_Response(
140                         [
141                                 'success' => true,
142                                 'data'    => 'Connected successfully.',
143                         ],
144                         200
145                 );
146         }
147
148         /**
149          * Verify user token.
150          *
151          * @return  array|WP_Error $response Response.
152          */
153         public static function verify_user_token() {
154                 $args     = [
155                         'body'      => [
156                                 'token'      => SaasApiToken::get(),
157                                 'saas-token' => SaasApiToken::get(),
158                                 'base_url'   => str_replace( '/wp-json/', '', get_rest_url() ),
159                         ],
160                         'sslverify' => false,
161                         'timeout'   => 60, //phpcs:ignore
WordPressVIPMinimum.Performance.RemoteRequestTimeout.timeout_timeout
162                 ];
163                 $response = wp_remote_post( SURE_TRIGGERS_API_SERVER_URL . '/token/verify', $args );
164
165                 return $response;
166         }
167
168         /**
169          * Verify connection.
170          *
171          * @return  array|WP_Error $response Response.
172          */
```

```
Line
173         public static function suretriggers_verify_wp_connection() {
174                 $args     = [
175                         'body'      => [
176                                 'saas-token'     => SaasApiToken::get(),
177                                 'base_url'       => str_replace( '/wp-json/', '', get_rest_url() ),
178                                 'plugin_version' => SURE_TRIGGERS_VER,
179                         ],
180                         'sslverify' => false,
181                         'timeout'   => 60, //phpcs:ignore WordPressVIPMinimum.Performance.RemoteRequestTimeout.timeout_timeout
182                 ];
183 $response = wp_remote_post( SURE_TRIGGERS_API_SERVER_URL . '/connection/wordpress/ping', $args );
184                 return $response;
185         }
186
187         /**
188          * Authenticate User for API calls.
189          *
190          * @param array|object $user USer.
191          *
192          * @return int|null
193          */
194         public function basic_auth_handler( $user ) {
195                 // Don't authenticate twice.
196                 if ( ! empty( $user ) ) {
197                         return $user;
198                 }
199
200                 // Check that we're trying to authenticate.
201                 if ( ! isset( $_SERVER['PHP_AUTH_USER'] ) || ! isset( $_SERVER['PHP_AUTH_PW'] ) ) { //phpcs:ignore
202                         return $user;
203                 }
204
205                 $username = sanitize_text_field( wp_unslash( $_SERVER['PHP_AUTH_USER'] ) ); //phpcs:ignore
206                 $password = sanitize_text_field( wp_unslash( $_SERVER['PHP_AUTH_PW'] ) ); //phpcs:ignore
207
208                 /**
209                  * In multi-site, wp_authenticate_spam_check filter is run on authentication. This filter calls.
210                  * get_currentuserinfo which in turn calls the determine_current_user filter. This leads to infinite.
211                  * recursion and a stack overflow unless the current function is removed from the determine_current_user.
212                  * filter during authentication.
213                  */
214                 remove_filter( 'determine_current_user', [ $this, 'basic_auth_handler' ], 20 );
215
216                 $user = wp_authenticate( $username, $password );
217
218                 add_filter( 'determine_current_user', [ $this, 'basic_auth_handler' ], 20 );
219
220                 if ( is_wp_error( $user ) ) {
221                         return null;
222                 }
223
224                 return $user->ID;
225         }
226
227         /**
228          * Authenticate user for new connection create api.
229          *
230          * @return bool
231          */
```

```
Line
232         public function is_current_user() {
233                 if ( current_user_can( 'manage_options' ) ) {
234                         return true;
235                 }
236                 return false;
237         }
238
239         /**
240          * Execute action events.
241          *
242          * @param WP_REST_Request $request Request data.
243          * @return WP_REST_Response
244          */
245         public function run_action( $request ) {
246                 $request->get_param( 'wp_user_id' );
247
248                 $user_id          = $request->get_param( 'wp_user_id' );
249                 $automation_id    = $request->get_param( 'automation_id' );
250                 $integration      = $request->get_param( 'integration' );
251                 $action_type      = $request->get_param( 'type_event' );
252                 $selected_options = $request->get_param( 'selected_options' );
253                 $context          = $request->get_param( 'context' );
254                 $fields           = $request->get_param( 'fields' );
255
256                 if ( empty( $user_id ) ) {
257                         $user_id = isset( $context['pluggable_data']
['wp_user_id'] ) ? sanitize_text_field( $context['pluggable_data']['wp_user_id'] ) : '';
258                 }
259
260                 if ( empty( $integration ) || empty( $action_type ) ) {
261                         return self::error_message( 'Integration or action type is missing' );
262                 }
263
264
if ( isset( $selected_options['wp_user_email'] ) && ! ( 'EDD' === $integration && 'find_user_purchased_download
' === $action_type ) ) {
265                         $is_valid = WordPress::validate_email( $selected_options['wp_user_email'] );
266
267                         if ( ! $is_valid->valid ) {
268                                 if ( $is_valid->multiple ) {
269                                         return self::error_message( 'One or more email address is not
valid.' );
270                                 } else {
271                                         return self::error_message( 'Email address is not valid.' );
272                                 }
273                         }
274
275                         if ( str_contains( $selected_options['wp_user_email'], ',' ) ) {
276                                 $email_list = explode( ',', $selected_options['wp_user_email'] );
277
278                                 foreach ( $email_list as $single_email ) {
279                                         if ( ! email_exists( trim( $single_email ) ) ) {
280                                                 return self::error_message( 'User with email
' . $single_email . ' does not exists.' );
281                                         }
282                                 }
283                         } else {
284                                 if ( ! email_exists( $selected_options['wp_user_email'] ) ) {
285                                         return self::error_message( 'User with email
' . $selected_options['wp_user_email'] . ' does not exists.' );
286                                 }
287                         }
288                 }
289                 $registered_actions = EventController::get_instance()->actions;
290                 $action_event       = $registered_actions[ $integration ][ $action_type ];
```

```
Line
291
292                 $fun_params = [
293                         $user_id,
294                         $automation_id,
295                         $fields,
296                         $selected_options,
297                         $context,
298                 ];
299
300                 try {
301                         $result = call_user_func_array(
302                                 $action_event['function'],
303                                 $fun_params
304                         );
305                         return self::success_message( $result );
306                 } catch ( Exception $e ) {
307                         return self::error_message( $e->getMessage(), 400 );
308                 }
309         }
310
311         /**
312          * Error message format.
313          *
314          * @param string $message Error message.
315          * @param string $status Error message.
316          *
317          * @return object
318          */
319         public static function error_message( $message, $status = 401 ) {
320                 return new WP_REST_Response(
321                         [
322                                 'success' => false,
323                                 'data'    => [
324                                         'errors' => $message,
325                                 ],
326                         ],
327                         $status
328                 );
329         }
330
331         /**
332          * Success message format.
333          *
334          * @param array $data response data to be sent.
335          *
336          * @return object
337          */
338         public static function success_message( $data = [] ) {
339                 $result = [];
340
341                 if ( ! empty( $data ) ) {
342                         $result['result'] = $data;
343                 }
344
345                 return new WP_REST_Response(
346                         [
347                                 'success' => true,
348                                 'data'    => $result,
349                         ],
350                         200
351                 );
352
353         }
```

```
354
355         /**
356          * Add/Remove/Update the triggers..
357          * When new/update/remove automation on Sass then execute this endpoint to update the automation.
358          *
359          * @param WP_REST_Request $request Request data.
360          * @return WP_REST_Response
361          */
362         public function manage_triggers( $request ) {
363                 $events = $request->get_param( 'events' ) ? json_decode( stripslashes( $request->get_param( 'events' ) ), true ) : [];
364
365                 // Selected field data from the trigger.
366                 $data = $request->get_param( 'data' ) ? json_decode( stripslashes( $request->get_param( 'data' ) ), true ) : [];
367
368                 // Get the trigger data from the option and append data in trigger data option.
369                 $trigger_data = OptionController::get_option( 'trigger_data' );
370                 if ( empty( $trigger_data ) ) {
371                         $trigger_data = [];
372                 }
373
374                 if ( is_array( $data ) && is_array( $events ) ) {
375                         $index = array_search( $data['trigger'], array_column( $events, 'trigger' ) );
376 if ( is_array( $trigger_data ) && false !== $index && $data['integration'] === $events[ $index ]['integration'] ) {
377                                 $trigger_data[ $data['integration'] ][ $data['trigger'] ]['selected_options'] = $data['selected_data'];
378                         }
379                 }
380
381                 OptionController::set_option( 'triggers', $events );
382                 // Set the new option for the trigger data.
383                 OptionController::set_option( 'trigger_data', $trigger_data );
384                 $events = array_column( $events, 'trigger' );
385                 return self::success_message(
386                         [
387                                 'events' => $events,
388                                 'data'   => $trigger_data,
389                         ]
390                 );
391         }
392
393         /**
394          * Send response to Saas that trigger is executed.
395          *
396          * @param array $trigger_data Trigger data.
397          *
398          * @return bool
399          */
400         public function trigger_listener( $trigger_data ) {
401                 // Pass unique WordPress webhook id.
402                 $wordpress_webhook_uuid                = str_replace( '-', '', wp_generate_uuid4() );
403                 $site_url                              = esc_url_raw( str_replace( '/wp-json/', '', get_site_url() ) );
404                 $site_url                              = preg_replace( '/^https?:\/\//', '', $site_url );
405                 $encoded_site_url                      = urlencode( (string) $site_url );
406                 $trigger_data['wordpress_webhook_uuid'] = $wordpress_webhook_uuid . '_' . $encoded_site_url;
407                 $args                                  = [
408                         'headers'   => [
409                                 'Authorization'  => 'Bearer ' . $this->secret_key,
410                                 'Referer'        => str_replace( '/wp-json/', '', get_site_url() ),
411                                 'RefererRestUrl' => str_replace( '/wp-json/', '', get_rest_url() ),
412                         ],
```

```
413                         'body'      => json_decode( wp_json_encode( $trigger_data ), 1 ),
414                         'sslverify' => false,
415                         'timeout'   => 60, //phpcs:ignore WordPressVIPMinimum.Performance.RemoteRequestTimeout.timeout_timeout
416                 ];
417
418                 /**
419                  *
420                  * Ignore line
421                  *
422                  * @phpstan-ignore-next-line
423                  */
424                 $response = wp_remote_post( SURE_TRIGGERS_WEBHOOK_SERVER_URL . '/wordpress/webhook', $args );
425                 // Store every webhook requests.
426                 $error_info = wp_remote_retrieve_body( $response );
427                 if ( 405 === wp_remote_retrieve_response_code( $response ) ) {
428                         $error_info = wp_remote_retrieve_response_message( $response );
429                 }
430                 if ( 0 === wp_remote_retrieve_response_code( $response ) ) {
431                         $error_info = __( 'Service not available', 'suretriggers' );
432                 }
433                 unset( $args['headers']['Authorization'] );
434
WebhookRequestsController::suretriggers_log_request( (string) wp_json_encode( $args ), (int) wp_remote_retrieve_response_code( $response ), $error_info );
435
436                 if ( wp_remote_retrieve_response_code( $response ) === 200 ) {
437                         return true;
438                 }
439
440                 return false;
441         }
442
443         /**
444          * Update the connection from SAAS.
445          *
446          * @param WP_REST_Request $request Request data.
447          *
448          * @return void
449          */
450         public function connection_update( $request ) {
451                 $secret = $request->get_param( 'secret_key' );
452                 if ( $secret && is_string( $secret ) ) {
453                         SaasApiToken::save( $secret );
454                 }
455         }
456
457         /**
458          * Disconnect connection
459          *
460          * @param WP_REST_Request $request Request data.
461          * @return WP_REST_Response
462          */
463         public function connection_disconnect( $request ) {
464                 SaasApiToken::save( null );
465                 return self::success_message();
466         }
467
468         /**
469          * Test Trigger
470          * When test trigger is initiated on Sass then execute this endpoint to create a transient for identifying trigger event.
471          *
472          * @param WP_REST_Request $request Request data.
```

```
473         * @return WP_REST_Response
474         */
475        public function test_triggers( $request ) {
476                $test_triggers = (array) OptionController::get_option( 'test_triggers', [] );
477                $event         = [
478                        'trigger'     => $request->get_param( 'trigger' ),
479                        'integration' => $request->get_param( 'integration' ),
480                ];
481
482                // if request is to delete the transient, delete it and return.
483                if ( $request->get_param( 'clear_transient_data' ) === 'yes' ) {
484                        $test_triggers = array_filter(
485                                $test_triggers,
486                                function ( $v ) use ( $event ) {
487                                        return $v !== $event;
488                                }
489                        );
490                        OptionController::set_option( 'test_triggers', $test_triggers );
491
492                        return;
493                }
494
495                $test_triggers[]   = $event;
496                $test_triggers     = array_unique( $test_triggers, SORT_REGULAR );
497                $tmp_test_triggers = [];
498
499                foreach ( $test_triggers as $test_trigger ) {
500                        if ( ! empty( $test_trigger['trigger'] ) ) {
501                                $tmp_test_triggers[] = $test_trigger;
502                        }
503                }
504
505                OptionController::set_option( 'test_triggers', $tmp_test_triggers );
506        }
507
508        /**
509         * OttoKit Connection Info
510         *
511         * @param array $debug_info Info data.
512         * @return array
513         */
514        public function sure_triggers_connection_info( $debug_info ) {
515                // Verify if OttoKit is connected successfully.
516                $response   = self::verify_user_token();
517                $connection = ( wp_remote_retrieve_response_code( $response ) === 200 );
518                if ( $connection ) {
519                        $connection_status = 'Connection Successfully Set';
520                } else {
521                        $connection_status = 'Error in Connection';
522                }
523                $debug_info['suretriggers'] = [
524                        'label'  => __( 'OttoKit', 'suretriggers' ),
525                        'fields' => [
526                                'suretriggers_status'  => [
527                                        'label'   => __( 'OttoKit Status', 'suretriggers' ),
528                                        'value'   => $connection_status,
529                                        'private' => false,
530                                ],
531                                'rest_url'             => [
532                                        'label'   => __( 'Rest URL', 'suretriggers' ),
533                                        'value'   => esc_url( get_rest_url() ),
534                                        'private' => false,
535                                ],
```

Line	
536	'suretriggers_version' => [
537	'label' => __('OttoKit Version', 'suretriggers'),
538	'value' => SURE_TRIGGERS_VER,
539	'private' => false,
540	],
541	],
542	];
543	return $debug_info;
544	}
545	
546	}
547	
548	RestController::get_instance();

Matt

Privacy

Public Code