

```

1  <?php /** @noinspection PhpUndefinedFieldInspection */
2  /**
3   * @package GNUTalerPayment
4   */
5  /**
6   * Plugin Name: GNU Taler Payment for Woocommerce
7   * Plugin URI: https://github.com/Sakaeo/GNU-Taler-Plugin
8   * //Or Wordpress pluin URI
9   * Description: This plugin enables the payment via the GNU Taler payment system
10  * Version: 1.0
11  * Author: Hofmann Dominique & Strübin Jan
12  * Author URI:
13  *
14  * License: GNU General Public License v3.0
15  * License URI: http://www.gnu.org/licenses/gpl-3.0.html
16  * WC requires at least: 2.2
17  */
18
19  /*
20   This program is free software: you can redistribute it and/or modify
21   it under the terms of the GNU General Public License as published by
22   the Free Software Foundation, either version 3 of the License, or
23   (at your option) any later version.
24
25   This program is distributed in the hope that it will be useful,
26   but WITHOUT ANY WARRANTY; without even the implied warranty of
27   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
28   GNU General Public License for more details.
29
30   You should have received a copy of the GNU General Public License
31   along with this program. If not, see <https://www.gnu.org/licenses/>.
32  */
33
34
35  require_once ABSPATH . 'wp-admin/includes/plugin.php';
36
37  //Exit if accessed directly.
38  if ( ! defined( 'ABSPATH' ) ) {
39      exit();
40  }
41
42  /*
43   * This action hook registers our PHP class as a WooCommerce payment gateway
44   */
45
46  /**
47   * Adds the GNU Taler payment method to the other payment gateways.
48   *
49   * @param $gateways - Array of all the payment gateways.
50   * @return array
51   * @since 1.0
52   */
53
54
55  function gnutaler_add_gateway_class( $gateways ) {
56      $gateways[] = 'WC_GNUTaler_Gateway';
57      return $gateways;
58  }
59
60  add_filter( 'woocommerce_payment_gateways', 'gnutaler_add_gateway_class' );
61
62  /**
63   * The class itself, please note that it is inside plugins_loaded action hook
64   */
65
66  add_action( 'plugins_loaded', 'gnutaler_init_gateway_class' );
67
68
69  function gnutaler_init_gateway_class()
70  {
71      //Check if WooCommerce is active, if not then deactivate and show error message
72      if ( ! in_array( 'woocommerce/woocommerce.php', apply_filters('active_plugins',

```

```

73     get_option( 'active_plugins' ) ), true ) ) {
74         deactivate_plugins( plugin_basename( __FILE__ ) );
75         wp_die( "<strong>GNU Taler</strong> requires <strong>WooCommerce</strong>
76         plugin to work normally. Please activate it or install it from <a
77         href=\"http://wordpress.org/plugins/woocommerce/\"
78         target=\"_blank\">here</a>.<br /><br />Back to the WordPress <a href=\"
79         get_admin_url( null, 'plugins.php' ) . \">Plugins page</a>.\" );
80     }
81
82     /**
83     * GNU Taler Payment Gateway class.
84     *
85     * Handles the payments from the Woocommerce Webshop and sends the transactions
86     to the GNU Taler Backend and the GNU Taler Wallet.
87     *
88     * @since 1.0
89     */
90     class WC_GNUTaler_Gateway extends WC_Payment_Gateway
91     {
92         /**
93         * Class constructor
94         */
95         public function __construct()
96         {
97             $this->id = 'gnutaler'; // payment gateway plugin ID
98             $this->icon = ''; // URL of the icon that will be displayed on checkout
99             page near the gateway name
100             $this->has_fields = false; // There is no need for custom checkout
101             fields, therefore set false
102             $this->method_title = 'GNU Taler Gateway';
103             $this->method_description = 'This plugin enables the payment via the GNU
104             Taler payment system'; // will be displayed on the options page
105
106             // gateways can support refunds, saved payment methods,
107             $this->supports = array(
108                 'products', 'refunds',
109             );
110             // Method with all the options fields
111             $this->init_form_fields();
112
113             // Load the settings.
114             $this->init_settings();
115             $this->title = $this->get_option( 'title' );
116             $this->description = $this->get_option( 'description' );
117             $this->enabled = $this->get_option( 'enabled' );
118             $this->GNU_Taler_Backend_URL = $this->get_option(
119             'GNU_Taler_Backend_URL' );
120             $this->GNU_Taler_Backend_API_Key = $this->get_option(
121             'GNU_Taler_Backend_API_Key' );
122             $this->Fulfillment_url = $this->get_option( 'Fulfillment_url' );
123             $this->Order_text = $this->get_option( 'Order_text' );
124             $this->merchant_information = $this->get_option( 'merchant_information' );
125             $this->merchant_name = $this->get_option( 'merchant_name' );
126
127             //Here we add the Javascript files to the webserver
128             add_action( 'woocommerce_before_checkout_form', static function () {
129                 wp_enqueue_script( 'taler-wallet-lib', plugin_dir_url( __FILE__ ) .
130                 'js/taler-wallet-lib.js' );
131             } );
132             add_action( 'woocommerce_before_checkout_form', static function () {
133                 wp_enqueue_script( 'WalletDetection', plugin_dir_url( __FILE__ ) .
134                 'js/WalletDetection.js' );
135             } );
136
137             // This action hook saves the settings
138             add_action( 'woocommerce_update_options_payment_gateways_' . $this->id,
139             array( $this, 'process_admin_options' ) );
140         }
141
142         /**
143         * Plugin options
144         */

```

```

131 public function init_form_fields()
132 {
133     $this->form_fields = array(
134         'enabled' => array(
135             'title' => 'Enable/Disable',
136             'label' => 'Enable GNU Taler Gateway',
137             'type' => 'checkbox',
138             'description' => '',
139             'default' => 'no',
140         ),
141         'title' => array(
142             'title' => 'Title',
143             'type' => 'text',
144             'description' => 'What name should the payment method have when
the costumer can choose how to pay .',
145             'default' => 'GNU Taler',
146             'desc_tip' => true,
147         ),
148         'description' => array(
149             'title' => 'Description',
150             'type' => 'textarea',
151             'description' => 'This controls the description which the
customer sees during checkout.',
152             'default' => 'Pay with the new Payment method GNU Taler.',
153         ),
154         'GNU_Taler_Backend_URL' => array(
155             'title' => 'GNU Taler Backend URL',
156             'type' => 'text',
157             'description' => 'Set the URL of the GNU Taler Backend.',
158             'default' => 'https://backend.demo.taler.net',
159         ),
160         'GNU_Taler_Backend_API_Key' => array(
161             'title' => 'GNU Taler Backend API Key',
162             'type' => 'text',
163             'description' => 'Set the API-key for the Authorization with the
GNU Taler Backend.',
164             'default' => 'ApiKey sandbox',
165         ),
166         'Fulfillment_url' => array(
167             'title' => 'GNU Taler Fulfillment URL',
168             'type' => 'text',
169             'description' => 'Set the URL where the customer should return
after finishing the payment process.',
170             'default' => get_home_url(),
171         ),
172         'Order_text' => array(
173             'title' => 'Summarytext of the order',
174             'type' => 'text',
175             'description' => 'Set the text the customer should see as a
summary when he confirms the payment.',
176             'default' => 'Order',
177         ),
178         'merchant_information' => array(
179             'title' => 'Enable/Disable',
180             'label' => 'Enable sending your merchant information to the GNU
Taler Backend',
181             'type' => 'checkbox',
182             'description' => 'Do you want to send your merchant information
to the GNU Taler Backend via the transaction',
183             'default' => 'yes',
184         ),
185         'merchant_name' => array(
186             'title' => 'Name of the webshop',
187             'type' => 'text',
188             'description' => 'Set the name of the webshop that the customer
will see during the payment transaction.',
189             'default' => '',
190         ),
191     );
192 }
193
194 /**

```

```

195      * Sends a request to a url via HTTP.
196      *
197      * Sends a request to a GNU Taler Backend over HTTP and returns the result.
198      * The request can be sent as POST, GET, PUT or another method.
199      *
200      * @param $method - POST, GET, PUT or another method.
201      * @param $backend_url - URL to the GNU Taler Backend.
202      * @param $body - The content of the request.
203      * @param $purpose - What return value is to be expected.
204      * @return array The return array will either have the successful return
205      * value or a detailed error message.
206      * @since 1.0
207      */
208 public function call_api( $method, $backend_url, $body, $purpose ): array
209 {
210     //create_url
211     $url = $this->create_api_url( $backend_url, $purpose, $body );
212
213     //Initialize curl request
214     $curl = $this->curl_init( $method, $body, $url );
215
216     // EXECUTE:
217     $result = curl_exec( $curl );
218
219     //HTTP Status Error handling
220     $message_array = $this->curl_error_handling( $curl, $result );
221     curl_close( $curl );
222     return $message_array;
223 }
224
225 /**
226  * Checks if the return http code is a success and if not what kind of error
227  * status it is.
228  *
229  * If the request was successful an array will be returned with the boolean
230  * value true, the http code and the result of the response.
231  * If the request failed an array will be returned with the boolean value
232  * false, the http code and a detailed error message.
233  *
234  * @param $curl - Created curl request for error handling.
235  * @param $result - The response from the backend, that will be returned if
236  * the request was successful
237  * @return array - Array with a boolean, a http code and a message will be
238  * returned
239  * @since 1.0
240  */
241 public function curl_error_handling( $curl, $result ): array
242 {
243     $http_code = curl_getinfo( $curl, CURLINFO_HTTP_CODE );
244     if ( curl_error( $curl ) ) {
245         $error_msg = curl_error( $curl );
246     }
247     curl_close( $curl );
248     if ( isset( $error_msg ) ) {
249         return array(
250             'result' => false,
251             'http_code' => $http_code,
252             'message' => $error_msg,
253         );
254     }
255     if ( $http_code === 200 ) {
256         return array(
257             'result' => true,
258             'http_code' => $http_code,
259             'message' => $result,
260         );
261     }
262     if ( preg_match( '^(4[0-9]{2})$', $http_code ) ) {
263         switch ( $http_code ) {
264             case 400:

```

```

261         return array(
262             'result' => false,
263             'http_code' => $http_code,
264             'message' => 'request',
265         );
266         break;
267     case 401:
268         return array(
269             'result' => false,
270             'http_code' => $http_code,
271             'message' => 'Unauthorized',
272         );
273         break;
274     case 403:
275         return array(
276             'result' => false,
277             'http_code' => $http_code,
278             'message' => 'Forbidden',
279         );
280         break;
281     case 404:
282         return array(
283             'result' => false,
284             'http_code' => $http_code,
285             'message' => 'Page Not Found',
286         );
287         break;
288     default:
289         return array(
290             'result' => false,
291             'http_code' => $http_code,
292             'message' => '4xx Client Error',
293         );
294         break;
295     }
296 } elseif ( preg_match( '(5[0-9]{2})', $http_code ) ) {
297     switch ( $http_code ) {
298         case '500':
299             return array(
300                 'result' => false,
301                 'http_code' => $http_code,
302                 'message' => 'Internal Server Error',
303             );
304             break;
305         case '502':
306             return array(
307                 'result' => false,
308                 'http_code' => $http_code,
309                 'message' => 'Bad Gateway',
310             );
311             break;
312         case '503':
313             return array(
314                 'result' => false,
315                 'http_code' => $http_code,
316                 'message' => 'Service Unavailable',
317             );
318             break;
319         case '504':
320             return array(
321                 'result' => false,
322                 'http_code' => $http_code,
323                 'message' => 'Gateway Timeout',
324             );
325             break;
326         default:
327             return array(
328                 'result' => false,
329                 'http_code' => $http_code,
330                 'message' => '5xx Client Error',
331             );
332             break;

```

```

333     }
334     } else {
335         return array(
336             'result' => false,
337             'http_code' => $http_code,
338             'message' => 'http status error',
339         );
340     }
341 }
342
343 /**
344  * Initialises the curl request and sets some necessary options depending on
345  * the method.
346  *
347  * Depending of the method chosen different options for the curl request
348  * will be set.
349  * Not depending on the method the settings for a return value,
350  * Authorization and Content-Type are being set.
351  *
352  * @param $method - POST, GET, PUT or another method.
353  * @param $body - Content of the request.
354  * @param $url - URL where the request will be send
355  * @return false|resource - Either the configured curl request will be
356  * returned or false if an error appears.
357  * @since 1.0
358  */
359 public function curl_init( $method, $body, $url )
360 {
361     $curl = curl_init();
362     $apikey = $this->get_option( 'GNU_Taler_Backend_API_Key' );
363
364     switch ( $method ) {
365         case 'POST':
366             curl_setopt( $curl, CURLOPT_POST, 1 );
367             if ( $body ) {
368                 curl_setopt( $curl, CURLOPT_POSTFIELDS, $body );
369             }
370             break;
371         case 'PUT':
372             curl_setopt( $curl, CURLOPT_CUSTOMREQUEST, 'PUT' );
373             if ( $body ) {
374                 curl_setopt( $curl, CURLOPT_POSTFIELDS, $body );
375             }
376             break;
377         case 'GET':
378             curl_setopt( $curl, CURLOPT_VERBOSE, 1 );
379             break;
380         default:
381             curl_setopt( $curl, CURLOPT_CUSTOMREQUEST, $method );
382             break;
383     }
384
385     // OPTIONS:
386     curl_setopt( $curl, CURLOPT_URL, $url );
387     curl_setopt( $curl, CURLOPT_HTTPHEADER, array(
388         'Authorization: ' . $apikey,
389         'Content-Type: application/json',
390     ) );
391     curl_setopt( $curl, CURLOPT_RETURNTRANSFER, 1 );
392     curl_setopt( $curl, CURLOPT_HTTPAUTH, CURLAUTH_BASIC );
393
394     return $curl;
395 }
396
397 /**
398  * Creates the final url depending on the purpose.
399  *
400  * @param $url - URL where the request will be send.
401  * @param $purpose - What will be added to the url.
402  * @param $body - Content of the request.
403  * @return string - return the final url.
404  * @since 1.0

```

```

401     */
402     public function create_api_url ( $url, $purpose, $body ): string
403     {
404         if ( $purpose === 'create_order' ) {
405             return $url . '/order';
406         }
407         if ( $purpose === 'confirm_payment' ) {
408             return $url . '/check-payment?order_id=' . $body;
409         }
410         if ( $purpose === 'create_refund' ) {
411             return $url . '/refund';
412         }
413         return $url;
414     }
415
416     /**
417     * Verifying if the url to the backend given in the plugin options is valid
418     * or not.
419     *
420     * @param $url - URL to the backend
421     * @return bool - Returns if valid or not.
422     * @since 1.0
423     */
424     public function verify_backend_url( $url ): bool
425     {
426         $result = $this->call_api( 'GET', $url, '', '' );
427         if ( $result['result'] ){
428             return true;
429         }
430         return false;
431     }
432
433     /**
434     * Processes the payment after the checkout
435     *
436     * If the payment process finished successfully the user is being redirected
437     * to its GNU Taler Wallet.
438     * If an error occurs it returns void and throws an error.
439     *
440     * @param $order_id - ID of the order to get the Order from the WooCommerce
441     * Webshop
442     * @return array|void - Array with result => success and redirection url
443     * otherwise it returns void.
444     * @since 1.0
445     */
446     public function process_payment( $order_id )
447     {
448         // we need it to get any order details
449         $wc_order = wc_get_order( $order_id );
450
451         if ( is_user_logged_in() )
452         {
453             $user_id = WC()->customer->get_id();
454         }
455         else
456         {
457             $user_id = 'Guest';
458             $this->add_log_entry( 'transaction', 'The customer started a
459             transaction without login, therefore the userid is unknown.' );
460         }
461
462         // Gets the url of the backend from the WooCommerce Settings
463         $backend_url = $this->get_option( 'GNU_Taler_Backend_URL', 1 );
464
465         //Log entry that the customer started the payment process
466         $this->add_log_entry( 'transaction', 'Userid: ' . $user_id . ' -
467         Orderid: ' . $order_id . ' - User started the payment process.' );
468
469         if ( ! $this->verify_backend_url( $backend_url ) ) {
470             wc_add_notice( 'Something went wrong please contact the system
471             administrator of the webshop and send the following error: GNU Taler
472             backend url invalid', 'error' );
473         }
474     }
475 }

```

```

465         $this->add_log_entry( 'error', 'Userid: ' . $user_id . ' - Orderid:
         ' . $order_id . ' - Checkout process failed - Invalid backend url.' );
466         return;
467     }
468     $order_json = $this->convert_to_checkout_json( $order_id );
469
470     $this->add_log_entry( 'transaction', 'Userid: ' . $user_id . ' -
     Orderid: ' . $order_id . ' - Transaction request send to GNU Taler
     backend' );
471     $order_request = $this->send_order_request( $backend_url, $order_json,
     $user_id, $order_id );
472
473     if ( $order_request['boolean'] ) {
474         //Completes the order
475         $wc_order->payment_complete();
476
477         //Empties the shopping cart
478         WC()->cart->empty_cart();
479
480         //Returns that the payment process finished successfully and
         redirects the costumer to confirm the payment via GNU Taler
481         $this->add_log_entry( 'transaction', 'Userid: ' . $user_id . ' -
         Orderid: ' . $order_id . ' - Customer is being redirected to the
         payment confirmation site' );
482         return array(
483             'result' => 'success',
484             'redirect' => $order_request['url'],
485         );
486     }
487     wc_add_notice( 'There seems to be a problem with the payment process,
     please try again or send the following message to a system
     administrator: ' . $order_request['http_code'] . ' - ' .
     $order_request['error_message'] );
488     $this->add_log_entry( 'error', 'Userid: ' . $user_id . ' - Orderid: ' .
     $order_id . ' - An error occurred during the payment process - ' .
     $order_request['http_code'] . ' - ' . $order_request['error_message'] );
489     $wc_order->set_status( 'cancelled' );
490     return;
491 }
492
493 /**
494  * Sends the transaction to the GNU Taler Backend
495  *
496  * If the payment process finishes successfully it returns an array with the
     redirection url to the GNU Taler Wallet and a boolean value for error
     handling.
497  * If an error occurs it returns an array with a boolean value for error
     handling, the http status code and an error message.
498  *
499  * @param $backend_url - URL where the request will be sent.
500  * @param $json - JSON array with the data of the order for the backend.
501  * @param $user_id - User id for logging.
502  * @param $order_id - Order id for logging.
503  * @return array|void - Array with boolean true|false, url or error message
     with http status code.
504  * @since 1.0
505  */
506 public function send_order_request( $backend_url, $json, $user_id, $order_id
     ): array
507 {
508     // Send the POST-Request via CURL to the GNU Taler Backend
509     $order_confirmation = $this->call_api( 'POST', $backend_url,
     json_encode($json, JSON_UNESCAPED_SLASHES), 'create_order' );
510
511     $order_message = $order_confirmation['message'];
512     $order_boolean = $order_confirmation['result'];
513
514     if ( $order_boolean ) {
515         $order_confirmation_id = explode( '"', $order_message )[3];
516
517         // Send the final confirmation to execute the payment transaction to
         the GNU Taler Backend

```



```

518     $payment_confirmation = $this->call_api( 'GET', $backend_url,
519     $order_confirmation_id, 'confirm_payment' );
520     $payment_message = $payment_confirmation['message'];
521     $payment_boolean = $order_confirmation['result'];
522
523     if ( $payment_boolean ) {
524
525         //Here we check what kind of http code came back from the backend
526         $payment_confirmation_url = explode( '"', $payment_message )[3];
527         $this->add_log_entry( 'transaction', 'Userid: ' . $user_id . ' -
528         Orderid: ' . $order_id . ' - Successfully received redirect url
529         to wallet from GNU Taler Backend' );
530         return array(
531             'boolean' => true,
532             'url' => $payment_confirmation_url,
533         );
534     }
535     $this->add_log_entry( 'error', 'Userid: ' . $user_id . ' - Orderid: '
536     . $order_id . ' - An error occurred during the second request to
537     the GNU Taler backend - ' . $payment_confirmation['http_code'] . ' -
538     ' . $payment_message );
539     return array(
540         'boolean' => false,
541         'http_code' => $payment_confirmation['http_code'],
542         'error_message' => $payment_message,
543     );
544 }
545
546 /**
547  * Converts the order into a JSON format that can be send to the GNU Taler
548  * Backend.
549  *
550  *
551  * The amount of the refund request can, at the moment, only be refunded in
552  * the currency 'KUDOS', which is the currency that the GNU Taler Payment
553  * system uses.
554  * This will change in the future and therefore the following code must be
555  * replaced by the code below: 'amount' => 'KUDOS:2',
556  * 'amount' => $wc_order_currency . ':' . $wc_order_total_amount,
557  *
558  * @param $order_id - To get the order from the WooCommerce Webshop
559  * @return array - return the JSON Format.
560  * @since 1.0
561  */
562 public function convert_to_checkout_json( $order_id ): array
563 {
564     $wc_order = wc_get_order( $order_id );
565     $wc_order_total_amount = $wc_order->get_total();
566     $wc_order_currency = $wc_order->get_currency();
567     $wc_cart = WC()->cart->get_cart();
568     $wc_order_id = $wc_order->get_order_key() . '_' .
569     $wc_order->get_order_number();
570     $merchant_option = $this->get_option( 'merchant_information' );
571
572     $wc_order_products_array = $this->mutate_products_to_json_format(
573     $wc_cart, $wc_order_currency );
574
575     $wc_order_merchant = $this->mutate_merchant_information_to_json_format(
576     $merchant_option );
577
578     $order_json = array(
579         'order' => array(

```

```

574         //'amount' => $wc_order_currency . ':' . $wc_order_total_amount,
575         'amount' => 'KUDOS:5',
576         'summary' => 'Order of the following items:',
577         'fulfillment_url' => $this->get_option( 'Fulfillment_url',
get_home_url() ),
578         'order_id' => $wc_order_id,
579         'merchant' => $wc_order_merchant,
580         'products' => $wc_order_products_array,
581     ),
582 );
583     return $order_json;
584 }
585
586 /**
587  * Mutates the products in the cart into a format which can be included in a
JSON file.
588  *
589  * @param $wc_cart - The content of the WooCommerce Cart.
590  * @param $wc_order_currency - The currency the WooCommerce Webshop uses.
591  * @return array - Returns an array of products.
592  * @since 1.0
593  */
594 public function mutate_products_to_json_format( $wc_cart, $wc_order_currency
): array
595 {
596     $wc_order_products_array = array();
597     foreach ( $wc_cart as $product ) {
598         $wc_order_products_array[] = array(
599             'description' => 'Order of product: ' .
600             $product['data']->get_title(),
601             'quantity' => $product['quantity'],
602             'price' => $wc_order_currency . ':' .
603             $product['data']->get_price(),
604             'product_id' => $product['data']->get_id(),
605         );
606     }
607     return $wc_order_products_array;
608 }
609
610 /**
611  * Mutates the merchant information of the webshop into a format which can
be included in a JSON file.
612  *
613  * @param $merchant_option - If the webshop owner allows to send the backend
their information
614  * @return array - Returns an array of merchant information's.
615  * @since 1.0
616  */
617 public function mutate_merchant_information_to_json_format( $merchant_option
): array
618 {
619     $whitechar_encounter = false;
620     $store_address_street = '';
621     $store_address_streetNr = '';
622
623     // When the option is enabled the informations of the merchant will be
included in the transaction
624     if ( $merchant_option === 'yes' ) {
625         // The country/state
626         $store_raw_country = get_option( 'woocommerce_default_country' );
627         $split_country = explode( ':', $store_raw_country );
628
629         // Country and state separated:
630         $store_country = $split_country[0];
631         $store_state = $split_country[1];
632
633         //Streetname and number
634         $store_address = get_option( 'woocommerce_store_address' );
635         $store_address_inverted = strrev( $store_address );
636         $store_address_array = str_split( $store_address_inverted );
637
638         //Split the address into street and street number

```

```

637         foreach ( $store_address_array as $char ) {
638             if ( is_numeric( $char ) && ! $whitechar_encounter ) {
639                 $store_address_streetNr .= $char;
640             } elseif ( ctype_space( $char ) ) {
641                 $whitechar_encounter = true;
642             } else {
643                 $store_address_street .= $char;
644             }
645         }
646         $wc_order_merchant_location = array(
647             'country' => $store_country,
648             'state' => $store_state,
649             'city' => WC()->countries->get_base_city(),
650             'ZIP code' => WC()->countries->get_base_postcode(),
651             'street' => strrev( $store_address_street ),
652             'street number' => strrev( $store_address_streetNr ),
653         );
654         return array(
655             'address' => $wc_order_merchant_location,
656             'name' => $this->get_option( 'merchant_name' ),
657         );
658     }
659     return array();
660 }
661
662 /**
663  * Processes the refund transaction if requested by the system administrator
664  * of the webshop
665  *
666  * If the refund request is finished successfully it returns an refund url,
667  * which can be send to the customer to finish the refund transaction.
668  * If an error it will throw a WP_Error message and inform the system
669  * administrator.
670  *
671  * @param $order_id - Order id for logging.
672  * @param null $amount - Amount that is requested to be refunded.
673  * @param string $reason - Reason for the refund request.
674  * @return bool|WP_Error - Returns true or throws an WP_Error message in
675  * case of error.
676  * @since 1.0
677  */
678 public function process_refund( $order_id, $amount = null, $reason = '' )
679 {
680     $wc_order = wc_get_order( $order_id );
681     $refund_json = $this->convert_refund_to_json( $wc_order, $amount,
682     $reason );
683
684     $user_id = wc_get_order($order_id)->get_customer_id();
685     if ( $user_id === 0 ){
686         $user_id = 'Guest';
687     }
688
689     $this->add_log_entry( 'transaction', 'Userid: ' . $user_id . ' -
690     Orderid: ' . $order_id . ' - Refund process of order: ' . $order_id . '
691     started with the refunded amount: ' . $amount . ' ' .
692     $wc_order->get_currency() . ' and the reason: ' . $reason );
693
694     // Gets the url of the backend from the WooCommerce Settings
695     $backend_url = $this->get_option( 'GNU_Taler_Backend_URL' );
696
697     //Get the current status of the order
698     $wc_order_status = $wc_order->get_status();
699
700     //Checks if current status is already set as paid
701     if ( $wc_order_status === 'processing' || $wc_order_status === 'on hold'
702     || $wc_order_status === 'completed' ) {
703
704         $this->add_log_entry( 'transaction', 'Userid: ' . $user_id . ' -
705         Orderid: ' . $order_id . ' - Refund request sent to the GNU Taler
706         Backend' );
707         $refund_request = $this->send_refund_request( $backend_url,
708         $refund_json );
709     }
710 }

```

```

697
698         if( $refund_request['boolean'] ) {
699             //Set the status as refunded and post the link to confirm the
            refund process via the GNU Taler payment method
700             $wc_order->update_status( 'refunded' );
701             $wc_order->add_order_note( 'The refund process finished
            successfully, please send the following url to the customer via
            an email to confirm the refund transaction.' );
702             $wc_order->add_order_note( $refund_request['url'] );
703             $this->add_log_entry( 'transaction', 'Userid: ' . $user_id . ' -
            Orderid: ' . $order_id . ' - Successfully received refund
            redirect url from GNU Taler backend, customer can now refund the
            given amount.' );
704             return true;
705         }
706         $this->add_log_entry( 'error', 'Userid: ' . $user_id . ' - Orderid:
            ' . $order_id . ' - An error occurred during the refund process - '
            . $refund_request['http_code'] . ' - ' .
            $refund_request['error_message'] );
707         return new WP_Error( 'error', 'An error occurred during the refund
            process, please try again or send the following message to your
            system administrator: ' . $refund_request['http_code'] . ' - ' .
            $refund_request['error_message'] );
708     }
709     $this->add_log_entry( 'error', 'Userid: ' . $user_id . ' - Orderid: ' .
            $order_id . ' - The status of the order does not allow a refund' );
710     return new WP_Error( 'error', 'The status of the order does not allow
            for a refund.' );
711 }
712
713 /**
714  * Sends the refund transaction to the GNU Taler Backend
715  *
716  * If the refund process finishes successfully it returns a boolean value
            true and sends the system administrator the refund url for the customer.
717  * If an error occurs it returns a WP_Error which will be displayed .
718  *
719  * @param $backend_url - URL where the request will be sent.
720  * @param $json - JSON array with the data of the refund request for the
            backend.
721  * @return array|void - Array with boolean true|false, url or error message
            with http status code.
722  * @since 1.0
723  */
724 public function send_refund_request( $backend_url, $json ): array
725 {
726     $refund_url = '';
727     $refund_confirmation = $this->call_api( 'POST', $backend_url,
            json_encode($json, JSON_UNESCAPED_SLASHES), 'create_refund' );
728
729     $message = $refund_confirmation['message'];
730     $refund_boolean = $refund_confirmation['result'];
731
732     if ($refund_boolean) {
733         //Here we check what kind of http code came back from the backend
734         $refund_confirmation_array = explode(' ', $message);
735         foreach ( $refund_confirmation_array as $value ) {
736
737             //Looping through the return value and checking if
            "refund_redirect_url" can be found
738             if ( strpos( $value, 'refund_redirect_url' ) ) {
739                 $refund_url = explode( '"', $value )[3];
740             }
741         }
742         return array(
743             'boolean' => true,
744             'url' => $refund_url,
745         );
746     }
747     return array(
748         'boolean' => false,
749         'url' => $refund_url,

```

```

750         'http_code' => $refund_confirmation['http_code'],
751         'error_message' => $refund_confirmation['message'],
752     );
753 }
754
755 /**
756  * Converts the information of the refund request into a JSON format that
757  * can be send to the GNU Taler Backend.
758  *
759  * The amount of the refund request can, at the moment, only be refunded in
760  * the currency 'KUDOS', which is the currency that the GNU Taler Payment
761  * system uses.
762  * This will change in the future and therefore the following code must be
763  * replaced by the code below: 'refund' => 'KUDOS:1'
764  * 'refund' => $order->get_currency() . ':' . $amount,
765  *
766  * @param $order - Order where the refund request originated from.
767  * @param $amount - Amount to be refunded.
768  * @param $reason - Reason of refund.
769  * @return array - returns the JSON Format.
770  * @since 1.0
771  */
772 public function convert_refund_to_json( $order, $amount, $reason ): array
773 {
774     return array(
775         'order_id' => $order->get_order_key() . '_' .
776         $order->get_order_number(),
777         'refund' => 'KUDOS:4',
778         //'refund' => $order->get_currency() . ':' . $amount,
779         'instance' => 'default',
780         'reason' => $reason,
781     );
782 }
783
784 /**
785  *
786  * Creates or opens the log files and writes a log entry.
787  *
788  * @param $type - What kind of log it is.
789  * @param $message - What the message of the log entry is.
790  * @return void - Returns void.
791  * @since 1.0
792  */
793 public function add_log_entry( $type, $message ): void
794 {
795     $file = null;
796     $timestamp = date( 'r' );
797     if ( $type === 'error' ) {
798         $file = fopen( __DIR__ . '/log/GNUTaler_Error.log', 'ab' );
799     }
800     elseif ( $type === 'transaction' ) {
801         $file = fopen( __DIR__ . '/log/GNUTaler_User_Transactions.log', 'ab'
802         );
803     }
804     else
805     {
806         $file = fopen( __DIR__ . '/log/GNUTaler_' . $type . '.log', 'ab' );
807     }
808     if ( $file !== null ){
809         fwrite( $file, $timestamp . ' - ' . $message . PHP_EOL );
810         fclose( $file );
811     }
812 }
813 }

```