```php
<?php /** @noinspection PhpUndefinedFieldInspection */
/**
 * @package GNUTalerPayment
 */
/**
 * Plugin Name: GNU Taler Payment for Woocommerce
 * Plugin URI: https://github.com/Sakaeo/GNU-Taler-Plugin
 *      //Or Wordpress pluin URI
 * Description: This plugin enables the payment via the GNU Taler payment system
 * Version: 1.0
 * Author: Hofmann Dominique & Strübin Jan
 * Author URI:
 *
 * License:           GNU General Public License v3.0
 * License URI:       http://www.gnu.org/licenses/gpl-3.0.html
 * WC requires at least: 2.2
 **/

/*
    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <https://www.gnu.org/licenses/>.
 */


require_once ABSPATH . 'wp-admin/includes/plugin.php';

//Exit if accessed directly.
if ( ! defined( 'ABSPATH' ) ) {
    exit();
}

/*
 * This action hook registers our PHP class as a WooCommerce payment gateway
 */

/**
 * Adds the GNU Taler payment method to the other payment gateways.
 *
 * @param $gateways - Array of all the payment gateways.
 * @return array
 * @since 1.0
 */


function gnutaler_add_gateway_class( $gateways ) {
    $gateways[] =   'WC_GNUTaler_Gateway';
    return $gateways;
}

add_filter( 'woocommerce_payment_gateways', 'gnutaler_add_gateway_class' );

/**
 * The class itself, please note that it is inside plugins_loaded action hook
 */

add_action( 'plugins_loaded', 'gnutaler_init_gateway_class' );


function gnutaler_init_gateway_class()
{
    //Check if WooCommerce is active, if not then deactivate and show error message
    if ( ! in_array( 'woocommerce/woocommerce.php', apply_filters('active_plugins',
        get_option('active_plugins' ) ), true ) ) {
            deactivate_plugins(plugin_basename(__FILE__));
            wp_die("<strong>GNU Taler</strong> requires <strong>WooCommerce</strong>
            plugin to work normally. Please activate it or install it from <a
            href=\"http://wordpress.org/plugins/woocommerce/\"
            target=\"_blank\">here</a>.<br /><br />Back to the WordPress <a href='" .
            get_admin_url(null, 'plugins.php') . "'>Plugins page</a>.");
        }

        /**
         * GNU Taler Payment Gateway class.
         *
         * Handles the payments from the Woocommerce Webshop and sends the transactions
         to the GNU Taler Backend and the GNU Taler Wallet.
         *
         * @since 1.0
         */
        class WC_GNUTaler_Gateway extends WC_Payment_Gateway
        {
            /**
             * Class constructor
             */
            public function __construct()
            {
                $this->id = 'gnutaler'; // payment gateway plugin ID
                $this->icon = ''; // URL of the icon that will be displayed on checkout
                page near the gateway name
                $this->has_fields = false; // There is no need for custom checkout
                fields, therefore set false
                $this->method_title = 'GNU Taler Gateway';
                $this->method_description = 'This plugin enables the payment via the GNU
                Taler payment system'; // will be displayed on the options page

                // gateways can support refunds, saved payment methods,
                $this->supports =   array(
                    'products', 'refunds',
                );
                // Method with all the options fields
                $this->init_form_fields();

                // Load the settings.
                $this->init_settings();
                $this->title = $this->get_option( 'title' );
                $this->description = $this->get_option( 'description' );
                $this->enabled = $this->get_option( 'enabled' );
                $this->GNU_Taler_Backend_URL = $this->get_option(
                'GNU_Taler_Backend_URL' );
                $this->GNU_Taler_Backend_API_Key = $this->get_option(
                'GNU_Taler_Backend_API_Key' );
                $this->Fulfillment_url = $this->get_option( 'Fulfillment_url' );
                $this->Order_text = $this->get_option( 'Order_text' );
                $this->merchant_information = $this->get_option( 'merchant_information' );
                $this->merchant_name = $this->get_option( 'merchant_name' );

                //Here we add the Javascript files to the webserver
                add_action( 'woocommerce_before_checkout_form', static function () {
                    wp_enqueue_script( 'taler-wallet-lib', plugin_dir_url( __FILE__ ) .
                    'js/taler-wallet-lib.js' );
                } );
                add_action( 'woocommerce_before_checkout_form', static function () {
                    wp_enqueue_script( 'WalletDetection', plugin_dir_url( __FILE__ ) .
                    'js/WalletDetection.js' );
                } );

                // This action hook saves the settings
                add_action( 'woocommerce_update_options_payment_gateways_' . $this->id,
                array( $this, 'process_admin_options' ) );
            }

            /**
             * Plugin options
             */
```

```php
        public function init_form_fields()
        {
            $this->form_fields  =   array(
                'enabled' => array(
                    'title' => 'Enable/Disable',
                    'label' => 'Enable GNU Taler Gateway',
                    'type' => 'checkbox',
                    'description' => '',
                    'default' => 'no',
                ),
                'title' =>  array(
                    'title' => 'Title',
                    'type' => 'text',
                    'description' => 'What name should the payment method have when the costumer can choose how to pay .',
                    'default' => 'GNU Taler',
                    'desc_tip' => true,
                ),
                'description' => array(
                    'title' => 'Description',
                    'type' => 'textarea',
                    'description' => 'This controls the description which the customer sees during checkout.',
                    'default' => 'Pay with the new Payment method GNU Taler.',
                ),
                'GNU_Taler_Backend_URL' => array(
                    'title' => 'GNU Taler Backend URL',
                    'type' => 'text',
                    'description' => 'Set the URL of the GNU Taler Backend.',
                    'default' => 'https://backend.demo.taler.net',
                ),
                'GNU_Taler_Backend_API_Key' => array(
                    'title' => 'GNU Taler Backend API Key',
                    'type' => 'text',
                    'description' => 'Set the API-key for the Authorization with the GNU Taler Backend.',
                    'default' => 'ApiKey sandbox',
                ),
                'Fulfillment_url' => array(
                    'title' => 'GNU Taler Fulfillment URL',
                    'type' => 'text',
                    'description' => 'Set the URL where the customer should return after finishing the payment process.',
                    'default' => get_home_url(),
                ),
                'Order_text' => array(
                    'title' => 'Summarytext of the order',
                    'type' => 'text',
                    'description' => 'Set the text the customer should see as a summary when he confirms the payment.',
                    'default' => 'Order',
                ),
                'merchant_information' => array(
                    'title' => 'Enable/Disable',
                    'label' => 'Enable sending your merchant information to the GNU Taler Backend',
                    'type' => 'checkbox',
                    'description' => 'Do you want to send your merchant information to the GNU Taler Backend via the transaction',
                    'default' => 'yes',
                ),
                'merchant_name' => array(
                    'title' => 'Name of the webshop',
                    'type' => 'text',
                    'description' => 'Set the name of the webshop that the customer will see during the payment transaction.',
                    'default' => '',
                ),
            );
        }

        /**
```

```php
         * Sends a request to a url via HTTP.
         *
         * Sends a request to a GNU Taler Backend over HTTP and returns the result.
         * The request can be sent as POST, GET, PUT or another method.
         *
         * @param $method - POST, GET, PUT or another method.
         * @param $backend_url - URL to the GNU Taler Backend.
         * @param $body - The content of the request.
         * @param $purpose - What return value is to be expected.
         * @return array The return array will either have the successful return value or a detailed error message.
         * @since 1.0
         *
         */
        public function call_api( $method, $backend_url, $body, $purpose ): array
        {
            //create_url
            $url = $this->create_api_url( $backend_url, $purpose, $body );

            //Initialize curl request
            $curl = $this->curl_init( $method, $body, $url );

            // EXECUTE:
            $result = curl_exec( $curl );

            //HTTP Status Error handling
            $message_array = $this->curl_error_handling( $curl, $result );
            curl_close( $curl );
            return $message_array;
        }

        /**
         * Checks if the return http code is a success and if not what kind of error status it is.
         *
         * If the request was successful an array will be returned with the boolean value true, the http code and the result of the response.
         * If the request failed an array will be returned with the boolean value false, the http code and a detailed error message.
         *
         * @param $curl - Created curl request for error handling.
         * @param $result - The response from the backend, that will be returned if the request was successful
         * @return array - Array with a boolean, a http code and a message will be returned
         * @since 1.0
         *
         */
        public function curl_error_handling( $curl, $result ): array
        {
            $http_code = curl_getinfo( $curl, CURLINFO_HTTP_CODE );
            if ( curl_error( $curl ) ) {
                $error_msg = curl_error( $curl );
            }
            curl_close( $curl );
            if ( isset( $error_msg ) ) {
                return array(
                    'result' => false,
                    'http_code' => $http_code,
                    'message' => $error_msg,
                );
            }
            if ( $http_code === 200 ) {
                return array(
                    'result' => true,
                    'http_code' => $http_code,
                    'message' => $result,
                );
            }
            if ( preg_match( '(4[0-9]{2})', $http_code ) ) {
                switch ($http_code) {
                    case 400:
                        return array(
                            'result' => false,
                            'http_code' => $http_code,
                            'message' => 'request',
                        );
                        break;
                    case 401:
                        return array(
                            'result' => false,
                            'http_code' => $http_code,
                            'message' => 'Unauthorized',
                        );
                        break;
                    case 403:
                        return array(
                            'result' => false,
                            'http_code' => $http_code,
                            'message' => 'Forbidden',
                        );
                        break;
                    case 404:
                        return array(
                            'result' => false,
                            'http_code' => $http_code,
                            'message' => 'Page Not Found',
                        );
                        break;
                    default:
                        return array(
                            'result' => false,
                            'http_code' => $http_code,
                            'message' => '4xx Client Error',
                        );
                        break;
                }
            } elseif ( preg_match( '(5[0-9]{2})', $http_code ) ) {
                switch ( $http_code ) {
                    case '500':
                        return array(
                            'result' => false,
                            'http_code' => $http_code,
                            'message' => 'Internal Server Error',
                        );
                        break;
                    case '502':
                        return array(
                            'result' => false,
                            'http_code' => $http_code,
                            'message' => 'Bad Gateway',
                        );
                        break;
                    case '503':
                        return array(
                            'result' => false,
                            'http_code' => $http_code,
                            'message' => 'Service Unavailable',
                        );
                        break;
                    case '504':
                        return array(
                            'result' => false,
                            'http_code' => $http_code,
                            'message' => 'Gateway Timeout',
                        );
                        break;
                    default:
                        return array(
                            'result' => false,
                            'http_code' => $http_code,
                            'message' => '5xx Client Error',
                        );
                        break;
```

```php
            }
        } else {
            return array(
                'result' => false,
                'http_code' => $http_code,
                'message' => 'http status error',
            );
        }
    }

    /**
     * Initialises the curl request and sets some necessary options depending on the method.
     *
     * Depending of the method chosen different options for the curl request will be set.
     * Not depending on the method the settings for a return value, Authorization and Content-Type are being set.
     *
     * @param $method - POST, GET, PUT or another method.
     * @param $body - Content of the request.
     * @param $url - URL where the request will be send
     * @return false|resource - Either the configured curl request will be returned or false if an error appears.
     * @since 1.0
     */
    public function curl_init( $method, $body, $url )
    {
        $curl = curl_init();
        $apikey = $this->get_option( 'GNU_Taler_Backend_API_Key' );

        switch ( $method ) {
            case 'POST':
                curl_setopt( $curl, CURLOPT_POST, 1 );
                if ( $body ) {
                    curl_setopt( $curl, CURLOPT_POSTFIELDS, $body );
                }
                break;
            case 'PUT':
                curl_setopt( $curl, CURLOPT_CUSTOMREQUEST, 'PUT' );
                if ( $body ) {
                    curl_setopt( $curl, CURLOPT_POSTFIELDS, $body );
                }
                break;
            case 'GET':
                curl_setopt( $curl, CURLOPT_VERBOSE, 1 );
                break;
            default:
                curl_setopt( $curl, CURLOPT_CUSTOMREQUEST, $method );
                break;
        }

        // OPTIONS:
        curl_setopt( $curl, CURLOPT_URL, $url );
        curl_setopt( $curl, CURLOPT_HTTPHEADER, array(
            'Authorization: ' . $apikey,
            'Content-Type: application/json',
        ) );
        curl_setopt( $curl, CURLOPT_RETURNTRANSFER, 1 );
        curl_setopt ($curl, CURLOPT_HTTPAUTH, CURLAUTH_BASIC );

        return $curl;
    }

    /**
     * Creates the final url depending on the purpose.
     *
     * @param $url - URL where the request will be send.
     * @param $purpose - What will be added to the url.
     * @param $body - Content of the request.
     * @return string - return the final url.
     * @since 1.0
```

```
         */
        public function create_api_url ($url, $purpose, $body ): string
        {
            if ( $purpose === 'create_order' ) {
                return $url . '/order';
            }
            if ( $purpose === 'confirm_payment' ) {
                return $url . '/check-payment?order_id=' . $body;
            }
            if ( $purpose === 'create_refund' ) {
                return $url . '/refund';
            }
            return $url;
        }

        /**
         * Verifying if the url to the backend given in the plugin options is valid or not.
         *
         * @param $url - URL to the backend
         * @return bool - Returns if valid or not.
         * @since 1.0
         */
        public function verify_backend_url( $url ): bool
        {
            $result = $this->call_api( 'GET', $url, '', '' );
            if ( $result['result'] ){
                return true;
            }
            return false;
        }

        /**
         * Processes the payment after the checkout
         *
         * If the payment process finished successfully the user is being redirected to its GNU Taler Wallet.
         * If an error occurs it returns void and throws an error.
         *
         * @param $order_id - ID of the order to get the Order from the WooCommerce Webshop
         * @return array|void - Array with result => success and redirection url otherwise it returns void.
         * @since 1.0
         */
        public function process_payment( $order_id )
        {
            // we need it to get any order detailes
            $wc_order = wc_get_order( $order_id );

            if ( is_user_logged_in() )
            {
                $user_id = WC()->customer->get_id();
            }
            else
            {
                $user_id = 'Guest';
                $this->add_log_entry( 'transaction', 'The customer started a transaction without login, therefore the userid is unknown.' );
            }

            // Gets the url of the backend from the WooCommerce Settings
            $backend_url = $this->get_option( 'GNU_Taler_Backend_URL', 1 );

            //Log entry that the customer started the payment process
            $this->add_log_entry( 'transaction', 'Userid: ' . $user_id .  ' - Orderid: ' . $order_id .  ' - User started the payment process.' );

            if ( ! $this->verify_backend_url( $backend_url ) ) {
                wc_add_notice( 'Something went wrong please contact the system administrator of the webshop and send the following error: GNU Taler backend url invalid', 'error' );
```

```php
            $this->add_log_entry( 'error', 'Userid: ' . $user_id . ' - Orderid: ' . $order_id . ' - Checkout process failed - Invalid backend url.' );
            return;
        }
        $order_json = $this->convert_to_checkout_json( $order_id );

        $this->add_log_entry( 'transaction', 'Userid: ' . $user_id . ' - Orderid: ' . $order_id . ' - Transaction request send to GNU Taler backend' );
        $order_request = $this->send_order_request( $backend_url, $order_json, $user_id, $order_id );

        if ( $order_request['boolean'] ) {
            //Completes the order
            $wc_order->payment_complete();

            //Empties the shopping cart
            WC()->cart->empty_cart();

            //Returns that the payment process finished successfully and redirects the costumer to confirm the payment via GNU Taler
            $this->add_log_entry( 'transaction', 'Userid: ' . $user_id . ' - Orderid: ' . $order_id . ' - Customer is being redirected to the payment confirmation site' );
            return array(
                'result' => 'success',
                'redirect' => $order_request['url'],
            );
        }
        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'] );
        $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'] );
        $wc_order->set_status( 'cancelled' );
        return;
    }

    /**
     * Sends the transaction to the GNU Taler Backend
     *
     * 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.
     * If an error occurs it returns an array with a boolean value for error handling, the http status code and an error message.
     *
     * @param $backend_url - URL where the request will be sent.
     * @param $json - JSON array with the data of the order for the backend.
     * @param $user_id - User id for logging.
     * @param $order_id - Order id for logging.
     * @return array|void - Array with boolean true|false, url or error message with http status code.
     * @since 1.0
     */
    public function send_order_request( $backend_url, $json, $user_id, $order_id ): array
    {
        // Send the POST-Request via CURL to the GNU Taler Backend
        $order_confirmation = $this->call_api( 'POST', $backend_url, json_encode($json, JSON_UNESCAPED_SLASHES), 'create_order' );

        $order_message = $order_confirmation['message'];
        $order_boolean = $order_confirmation['result'];

        if ( $order_boolean ) {
            $order_confirmation_id = explode( '"', $order_message )[3];

            // Send the final confirmation to execute the payment transaction to the GNU Taler Backend
            $payment_confirmation = $this->call_api( 'GET', $backend_url, $order_confirmation_id, 'confirm_payment' );
            $payment_message = $payment_confirmation['message'];
            $payment_boolean = $order_confirmation['result'];

            if ( $payment_boolean ) {

                //Here we check what kind of http code came back from the backend
                $payment_confirmation_url = explode( '"', $payment_message )[3];
                $this->add_log_entry( 'transaction', 'Userid: ' . $user_id . ' - Orderid: ' . $order_id . ' - Successfully received redirect url to wallet from GNU Taler Backend' );
                return array(
                    'boolean' => true,
                    'url' => $payment_confirmation_url,
                );
            }
            $this->add_log_entry( 'error', 'Userid: ' . $user_id .  ' - Orderid: ' . $order_id . ' - An error occurred during the second request to the GNU Taler backend - ' . $payment_confirmation['http_code'] . ' - ' . $payment_message );
            return array(
                'boolean' => false,
                'http_code' => $payment_confirmation['http_code'],
                'error_message' => $payment_message,
            );
        }
        $this->add_log_entry( 'error', 'Userid: ' . $user_id .  ' - Orderid: ' . $order_id . ' - An error occurred during the first request to the GNU Taler backend - ' . $order_confirmation['http_code'] . ' - ' . $order_message );
        return array(
            'boolean' => false,
            'http_code' => $order_confirmation['http_code'],
            'error_message' => $order_message,
        );
    }

    /**
     * Converts the order into a JSON format that can be send to the GNU Taler Backend.
     *
     *
     * The amount of the refund request can, at the moment, only be refunded in the currency 'KUDOS', which is the currency that the GNU Taler Payment system uses.
     * This will change in the future and therefore the following code must be replaced by the code below: 'amount' => 'KUDOS:2',
     * 'amount' => $wc_order_currency . ':' . $wc_order_total_amount,
     *         *
     * @param $order_id - To get the order from the WooCommerce Webshop
     * @return array - return the JSON Format.
     * @since 1.0
     */
    public function convert_to_checkout_json( $order_id ): array
    {
        $wc_order = wc_get_order( $order_id );
        $wc_order_total_amount = $wc_order->get_total();
        $wc_order_currency = $wc_order->get_currency();
        $wc_cart = WC()->cart->get_cart();
        $wc_order_id = $wc_order->get_order_key() . '_' . $wc_order->get_order_number();
        $merchant_option = $this->get_option( 'merchant_information' );

        $wc_order_products_array = $this->mutate_products_to_json_format( $wc_cart, $wc_order_currency );

        $wc_order_merchant = $this->mutate_merchant_information_to_json_format( $merchant_option );

        $order_json = array(
            'order' => array(
                //'amount' => $wc_order_currency . ':' . $wc_order_total_amount,
                'amount' => 'KUDOS:5',
                'summary' => 'Order of the following items:',
                'fulfillment_url' => $this->get_option( 'Fulfillment_url', get_home_url() ),
                'order_id' => $wc_order_id,
                'merchant' => $wc_order_merchant,
                'products' => $wc_order_products_array,
            ),
        );
        return $order_json;
    }

    /**
     * Mutates the products in the cart into a format which can be included in a JSON file.
     *
     * @param $wc_cart - The content of the WooCommerce Cart.
     * @param $wc_order_currency - The currency the WooCommerce Webshop uses.
     * @return array - Returns an array of products.
     * @since 1.0
     */
    public function mutate_products_to_json_format( $wc_cart, $wc_order_currency ): array
    {
        $wc_order_products_array = array();
        foreach ( $wc_cart as $product ) {
            $wc_order_products_array[] = array(
                'description' => 'Order of product: ' . $product['data']->get_title(),
                'quantity' => $product['quantity'],
                'price' => $wc_order_currency . ':' . $product['data']->get_price(),
                'product_id' => $product['data']->get_id(),
            );
        }
        return $wc_order_products_array;
    }

    /**
     * Mutates the merchant information of the webshop into a format which can be included in a JSON file.
     *
     * @param $merchant_option - If the webshop owner allows to send the backend their information
     * @return array - Returns an array of merchant information's.
     * @since 1.0
     */
    public function mutate_merchant_information_to_json_format( $merchant_option ): array
    {
        $whitechar_encounter = false;
        $store_address_street = '';
        $store_address_streetNr = '';

        // When the option is enabled the informations of the merchant will be included in the transaction
        if ( $merchant_option === 'yes' ) {
            // The country/state
            $store_raw_country = get_option( 'woocommerce_default_country' );
            $split_country = explode( ':', $store_raw_country );

            // Country and state separated:
            $store_country = $split_country[0];
            $store_state = $split_country[1];

            //Streetname and number
            $store_address = get_option( 'woocommerce_store_address' );
            $store_address_inverted = strrev( $store_address );
            $store_address_array = str_split( $store_address_inverted );

            //Split the address into street and street number
            foreach ( $store_address_array as $char ) {
                if ( is_numeric( $char ) && ! $whitechar_encounter ) {
                    $store_address_streetNr .= $char;
                } elseif ( ctype_space( $char ) ) {
                    $whitechar_encounter = true;
                } else {
                    $store_address_street .= $char;
                }
            }
            $wc_order_merchant_location = array(
                'country' => $store_country,
                'state' => $store_state,
                'city' => WC()->countries->get_base_city(),
                'ZIP code' => WC()->countries->get_base_postcode(),
                'street' => strrev( $store_address_street ),
                'street number' => strrev( $store_address_streetNr ),
            );
            return array(
                'address' => $wc_order_merchant_location,
                'name' => $this->get_option( 'merchant_name' ),
            );
        }
        return array();
    }

    /**
     * Processes the refund transaction if requested by the system administrator of the webshop
     *
     * If the refund request is finished successfully it returns an refund url, which can be send to the customer to finish the refund transaction.
     * If an error it will throw a WP_Error message and inform the system administrator.
     *
     * @param $order_id - Order id for logging.
     * @param null $amount - Amount that is requested to be refunded.
     * @param string $reason - Reason for the refund request.
     * @return bool|WP_Error - Returns true or throws an WP_Error message in case of error.
     * @since 1.0
     */
    public function process_refund( $order_id, $amount = null, $reason = '' )
    {
        $wc_order = wc_get_order( $order_id );
        $refund_json = $this->convert_refund_to_json( $wc_order, $amount, $reason );

        $user_id = wc_get_order($order_id)->get_customer_id();
        if ( $user_id === 0 ){
            $user_id = 'Guest';
        }

        $this->add_log_entry( 'transaction', 'Userid: ' . $user_id .  ' - Orderid: ' . $order_id . ' - Refund process of order: ' . $order_id . ' started with the refunded amount: ' . $amount . ' ' . $wc_order->get_currency() . ' and the reason: ' . $reason );

        // Gets the url of the backend from the WooCommerce Settings
        $backend_url = $this->get_option( 'GNU_Taler_Backend_URL' );

        //Get the current status of the order
        $wc_order_status = $wc_order->get_status();

        //Checks if current status is already set as paid
        if ( $wc_order_status === 'processing' || $wc_order_status === 'on hold' || $wc_order_status === 'completed' ) {

            $this->add_log_entry( 'transaction', 'Userid: ' . $user_id . ' - Orderid: ' . $order_id . ' - Refund request sent to the GNU Taler Backend' );
            $refund_request = $this->send_refund_request( $backend_url, $refund_json );

            if( $refund_request['boolean'] ) {
                //Set the status as refunded and post the link to confirm the refund process via the GNU Taler payment method
                $wc_order->update_status( 'refunded' );
                $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.' );
                $wc_order->add_order_note( $refund_request['url'] );
                $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.' );
                return true;
            }
            $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'] );
            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'] );
        }
        $this->add_log_entry( 'error', 'Userid: ' . $user_id . ' - Orderid: ' . $order_id . ' - The status of the order does not allow a refund' );
        return new WP_Error( 'error', 'The status of the order does not allow for a refund.' );
    }

    /**
     * Sends the refund transaction to the GNU Taler Backend
     *
     * If the refund process finishes successfully it returns a boolean value true and sends the system administrator the refund url for the customer.
     * If an error occurs it returns a WP_Error which will be displayed .
     *
     * @param $backend_url - URL where the request will be sent.
     * @param $json - JSON array with the data of the refund request for the backend.
     * @return array|void - Array with boolean true|false, url or error message with http status code.
     * @since 1.0
     */
    public function send_refund_request( $backend_url, $json ): array
    {
        $refund_url = '';
        $refund_confirmation = $this->call_api( 'POST', $backend_url, json_encode($json, JSON_UNESCAPED_SLASHES), 'create_refund' );

        $message = $refund_confirmation['message'];
        $refund_boolean = $refund_confirmation['result'];

        if ($refund_boolean) {
            //Here we check what kind of http code came back from the backend
            $refund_confirmation_array = explode(',', $message);
            foreach ( $refund_confirmation_array as $value ) {

                //Looping through the return value and checking if "refund_redirect_url" can be found
                if ( strpos( $value, 'refund_redirect_url' ) ) {
                    $refund_url = explode( '"', $value )[3];
                }
            }
            return array(
                'boolean' => true,
                'url' => $refund_url,
            );
        }
        return array(
            'boolean' => false,
            'url' => $refund_url,
                'http_code' => $refund_confirmation['http_code'],
                'error_message' => $refund_confirmation['message'],
            );
        }

        /**
         * Converts the information of the refund request into a JSON format that can be send to the GNU Taler Backend.
         *
         *  The amount of the refund request can, at the moment, only be refunded in the currency 'KUDOS', which is the currency that the GNU Taler Payment system uses.
         *  This will change in the future and therefore the following code must be replaced by the code below: 'refund' => 'KUDOS:1'
         * 'refund' => $order->get_currency() . ':' . $amount,
         *
         * @param $order - Order where the refund request originated from.
         * @param $amount - Amount to be refunded.
         * @param $reason - Reason of refund.
         * @return array - returns the JSON Format.
         * @since 1.0
         */
        public function convert_refund_to_json( $order, $amount, $reason ): array
        {
            return array(
                'order_id' => $order->get_order_key() . '_' . $order->get_order_number(),
                'refund' => 'KUDOS:4',
                //'refund' => $order->get_currency() . ':' . $amount,
                'instance' => 'default',
                'reason' => $reason,
            );
        }

        /**
         *
         * Creates or opens the log files and writes a log entry.
         *
         * @param $type - What kind of log it is.
         * @param $message - What the message of the log entry is.
         * @return void - Returns void.
         * @since 1.0
         */
        public function add_log_entry( $type, $message ): void
        {
            $file = null;
            $timestamp = date( 'r' );
            if ( $type === 'error' ) {
                $file = fopen( __DIR__ . '/log/GNUTaler_Error.log', 'ab' );
            }
            elseif ( $type === 'transaction' ) {
                $file = fopen( __DIR__ . '/log/GNUTaler_User_Transactions.log', 'ab' );
            }
            else
            {
                $file = fopen( __DIR__ . '/log/GNUTaler_' . $type . '.log', 'ab' );
            }
            if ( $file !== null ){
                fwrite( $file, $timestamp . ' - ' . $message . PHP_EOL );
                fclose( $file );
            }
        }
    }
}
```