Ta praca będzie poświęcona stworzeniu w pełni działającej wtyczki WordPress. Jej zadaniem będzie umożliwienie pełnej integracji pomiędzy systemem Księstwa Sarmacji a dowolną instalacją WP. W tym artykule prześledzimy, krok po kroku, proces tworzenia takiej wtyczki.

Czytelniku: jeżeli nie czytałeś, polecam wcześniejszą lekturę poprzedniego artykułu.

Repozytorium GIT

Pierwsza rzecz, która powinna być jasna dla każdego programisty to panowanie nad kodem. Wiele lat temu robiło się to w taki sposób, że nadawało się kolejne numery katalog, często dodawano plik z datą. W zasadzie zależało to tylko od inwencji twórcy danego rozwiązania. Które nie dosyć, że niewygodne i szybko zapominane, nie były ze sobą w żaden sposób kompatybilne. I rzadko można było cokolwiek zautomatyzować.

Na szczęście wraz z coraz większą popularnością komputerów, przyszły rozwiązania w postaci repozytoriów plików. Obecnie najpopularniejszymi rozwiązaniami są Git i SVN, w każdym razie jeśli chodzi o otwarte oprogramowanie. W naszej pracy posłużymy się repozytorium Git na platformie GitHub.com.

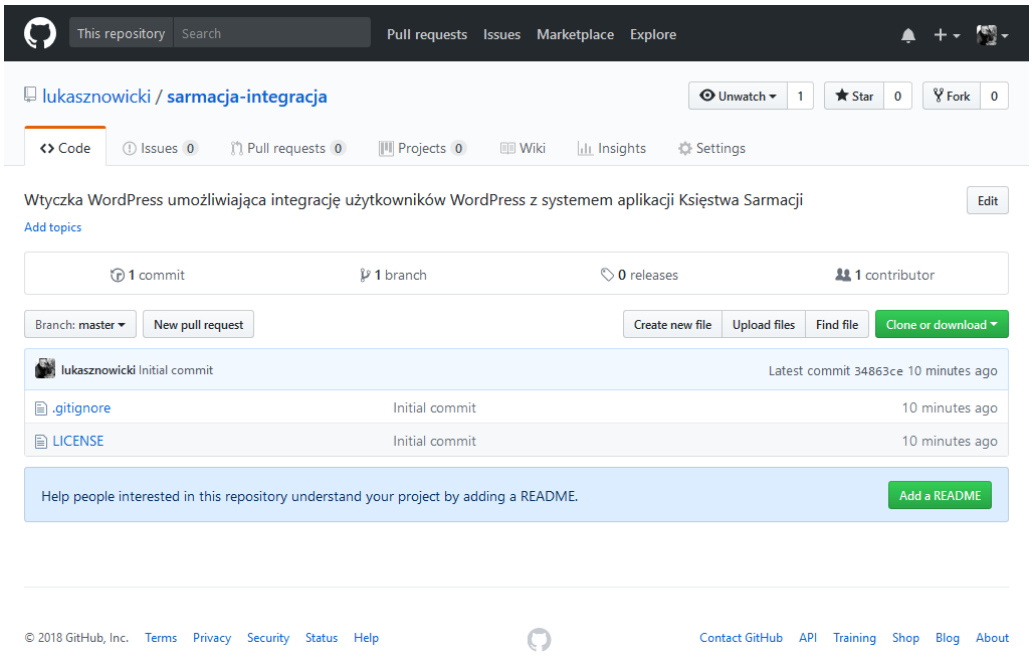
Adres WWW naszego repozytorium to:

https://github.com/lukasznowicki/sarmacja-integracja

Adres samego repozytorium do pobrania to:

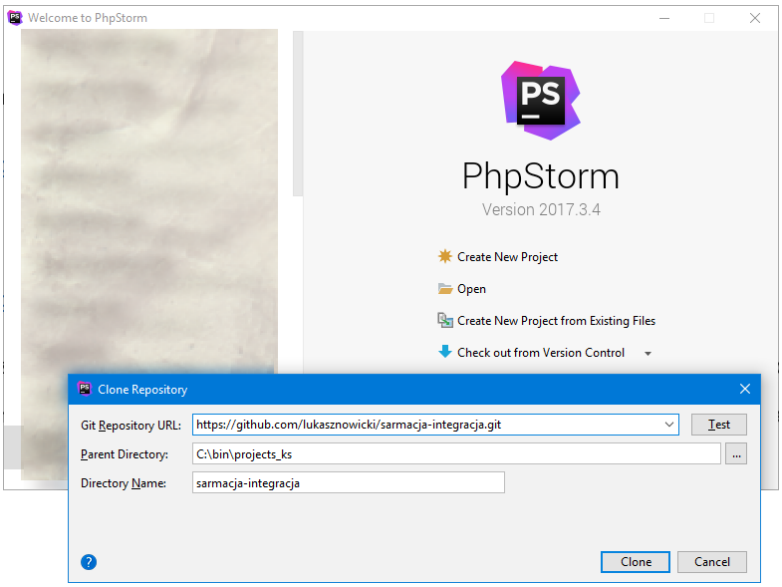
https://github.com/lukasznowicki/sarmacja-integracja.git

W trakcie tworzenia repozytorium, korzystając z narzędzi GitHub, od razu stworzyłem plik .gitignore, dzięki któremu można pomijać pliki / katalogi (np. te z konfiguracją) oraz plik licencji, wskazując licencję MIT. W tej chwili repozytorium wygląda tak:

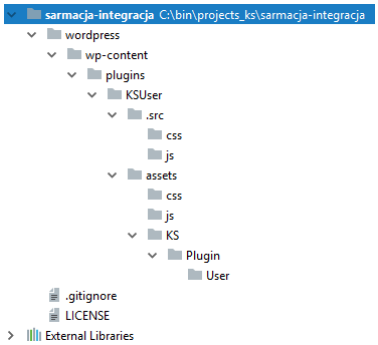

This repository
Search
Pull requests
Issues
Marketplace
Explore
lukasznowicki / sarmacja-integracja
Unwatch 1
Star 0
Fork 0
Code
Issues 0
Pull requests 0
Projects 0
Wiki
Insights
Settings
Wtyczka WordPress umożliwiająca integrację użytkowników WordPress z systemem aplikacji Księstwa Sarmacji
Edit
Add topics
1 commit
1 branch
0 releases
1 contributor
Branch: master
New pull request
Create new file
Upload files
Find file
Clone or download
lukasznowicki Initial commit
Latest commit 34863ce 10 minutes ago
.gitignore
Initial commit
10 minutes ago
LICENSE
Initial commit
10 minutes ago
Help people interested in this repository understand your project by adding a README.
Add a README
© 2018 GitHub, Inc.
Terms
Privacy
Security
Status
Help
Contact GitHub
API
Training
Shop
Blog
About

Projekt lokalnie

Następnie tworzę projekt na swoim dysku. Wykorzystam do tego katalog *C:\bin\projects_ks*. Do pisania kodu używam świetnego programu PhpStorm. Wybieram pobranie projektu z systemu kontroli wersji, wskazuję Git oraz wklejam adres repozytorium.



Po chwili jestem już w projekcie. Pierwsze co robię, to tworzę strukturę katalogów, która odpowiada strukturze WordPress-a. Dzięki temu, nikt nie będzie miał problemów z szybkim poznaniem struktury kodu. Po chwili mój projekt wygląda tak:



Tak rozbudowana struktura katalogów może się wydawać pewną, może nawet dużą przesadą. I tak jest w istocie. Jednak dzięki temu nasz kod nie będzie nam się plątał, mylił, mamy wszystko rozplanowane i na wszystko znajdzie się odpowiednie miejsce. Może się okazać, że w niedalekiej przyszłości trzeba będzie wtyczkę rozbudować.

Tutaj dosyć istotna uwaga. ***Kiedy realizuje się projekt informatyczny, należy pamiętać, że nie da się przewidzieć wszystkich jego przyszłych funkcjonalności i realizować różnego rodzaju zadań, zanim jeszcze będę potrzebne.*** To prawda. Jednak tutaj niczego nadmiarowo nie budujemy, tutaj tworzymy przejrzysty szkielet.

Pierwsze pliki

Zaczynamy standardowo, od głównego pliku wtyczki. Plik nazywa się tak samo jak katalog wtyczki, jest to też tożsame z nazwą wtyczki. Zatem do dzieła, pamiętając o tym że nie dodajemy informacji o tłumaczeniu – nasza wtyczka nie musi działać po angielsku. A jak kiedyś będzie musiała, to się doda. Tworzymy więc plik i już po chwili *KSUser.php* wygląda w ten sposób:

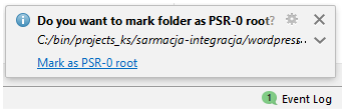
```php
<?php
/*
Plugin Name: KS User
Plugin URI: http://fc.sarmacja.org/viewtopic.php?f=726&t=24413
Description: Wtyczka umożliwiająca integrację systemów Księstwa Sarmacji z WP
Version: 0.1.1
Author: Ł.N.
Author URI: https://lukasznowicki.info/
License: MIT
License URI: https://raw.githubusercontent.com/lukasznowicki/sarmacja-integracja/master/LICENSE
*/
namespace KS\Plugin\User;

defined( name: 'ABSPATH') or exit;

```

Zwracam uwagę na linijkę, która sprawdza czy stała *ABSPATH* została zadeklarowana. Jeżeli nie, to plik jest wywoływany spoza WordPress i należy zakończyć jego wykonywanie.

Co dalej? Zawsze lubię sobie ułatwiać pracę, więc jeżeli w projekcie korzystam z więcej niż jednej klasy, używam systemu, który będzie sam ładował żądane klasy, zdejmując ze mnie konieczność wczytywania odpowiednich plików. W Internecie można znaleźć dużo przykładów, ja kiedyś któryś wybrałem, potem go sobie udoskonalałem w ramach potrzeb i ostatecznie wyszedł mi plik *Autoloader.php*, który zamieścimy w katalogu */KSUser/KS/Plugin/User/*.



Przy okazji, kiedy stworzyłem Autoloader w podkatalogu, którego struktura (rodzice) zgadzają się z przestrzenią nazw (namespace), program PhpStorm automatycznie zapyta mnie czy uznać folder nadrzędny za ścieżkę PSR-0, czyli przestarzały standard przetwarzania przestrzeni nazw na strukturę katalogu. Obecnie używa się PSR-4, który jest bardzo podobny a różnice pomiędzy nimi wykraczają poza tę pracę[i].

Po dodaniu pliku głównego i pliku odpowiedzialnego za automatyczne ładowanie, możemy wysłać całość na serwer GIT[ii]. Oczywiście można to zrobić z linii poleceń za pomocą komend *git add*, *git commit* oraz *git push*, ale na początek wygodniej będzie posłużyć się menu kontekstowym i odpowiednimi komendami git wybieranymi z menu.

Skoro mamy już sam początek wtyczki (nazwa, tego typu rzeczy) oraz klasę do automatycznego ładowania innych klas, warto by ją włączyć. W tym celu musimy…. Załadować klasę ręcznie. Niestety, tego nie przeszkodzimy, ale to tylko raz. Piszemy odpowiedni kod, który:

1. Zdefiniuje stałą PLUGIN_CLASS_DIR, która będzie wskazywała miejsce położenia „silnika" wtyczki,
2. załaduje plik Autoloader.php z klasą zapewniającą automatyczne ładowanie innych klas,
3. zainicjuje i skonfiguruje klasę Autoloader.

Po wykonaniu, kod wygląda następująco:

```php
namespace KS\Plugin\User;

defined( name: 'ABSPATH') or exit;

define( __NAMESPACE__ . '\PLUGIN_CLASS_DIR', trailingslashit( string: __DIR__ ) . 'KS' . \DIRECTORY_SEPARATOR . 'Plugin' . \DIRECTORY_SEPARATOR . 'User' . \DIRECTORY_SEPARATOR );

require_once PLUGIN_CLASS_DIR . 'Autoloader.php';

$ksuser_autoloader = new Autoloader();
$ksuser_autoloader->register();
$ksuser_autoloader->add_namespace( prefix: __NAMESPACE__, dir: PLUGIN_CLASS_DIR );
```

Teraz możemy przystąpić do dołączenia klasy Integracja do naszego projektu. Klasa dostępna jest na forum centralnym[iii], skąd pobieramy archiwum i je rozpakowujemy. W środku, wbrew temu co jest napisane na forum, nie znajdziemy pliku example.php tylko ja.php, ale proszę się tym nie zrażać. To jest również plik z przykładami. Nas w tej chwili interesuje jednak plik Integracja.php. Zaglądamy do środka i widzimy groźnie brzmiącą formułę o tym, aby nie ważyć się pliku modyfikować. To oczywiście słuszne, jednakże my sobie na malutką modyfikację pozwolimy – a mianowicie na samym początku pliku dopiszemy naszą przestrzeń nazw. Plik umieszczamy tam, gdzie Autoloader.php, w końcu to nasza nowa klasa. Moglibyśmy nie dopisywać przestrzeni

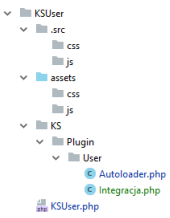
```php
<?php
namespace KS\Plugin\User;

//UPRASZA SIĘ O NIE ZMIENIANIE NICZEGO W TYM PLIKU

class Integracja {

    private $appId;
    private $appSecret;
    private $appName;
    private $user = null; //tu przetrzymujemy podstawowe dane usera
    private $wynik;
    private $accessToken = null;
    private $adress;
    private $options;
    private $config;
    private $sarmaUrl = 'http://www.sarmacja.org/integracja/';

    public function setConfiguration($config) {
        if (!session_id()) {
            session_start();
        }
```



nazw i pozostawić plik w katalogu głównym, jednakże to mogłoby kiedyś odbić nam się czkawką, gdyby inny twórca wtyczki lub szablonu postanowił użyć tej klasy. Lub o takiej samej nazwie.

Uwaga: plik Integracja.php został dodany do pliku .gitignore, który zezwala na ignorowanie plików, katalogów (zarówno pojedynczych jak i grup) w trakcie wysyłania kodu do repozytorium. Dlaczego? Ano, nie jestem pewien statusu pliku i nie wiem czy autorzy wyraziliby zgodę na takie rozpowszechnienie ich kodu. Stąd ten wyjątek. Tworząc swoją wtyczkę pamiętaj o tym – najważniejszym – pliku.

Teraz jesteśmy gotowi do pisania wtyczki!

Założenia

Wtyczka ma umożliwiać integrację systemu Księstwa Sarmacji z CMS WordPress. Dla osób, które nie są mieszkańcami Księstwa, nie posiadającymi własnego identyfikatora, rejestracja i oglądanie strony w ogóle nie powinno być możliwe. Użytkownik niezalogowany, musi się zidentyfikować. Proste? Proste.

Szkielet klasy wtyczki

W katalogu /KSUser/KS/Plugin/User/ tworzymy plik Plugin.php i umieszczamy w nim klasę Plugin. Teraz w pliku KSUser.php tworzymy odwołanie do naszej klasy:

```
new Plugin();
```

To wszystko. Klasa zostanie wywołana i choć na razie nic jeszcze nie zrobi – to plik KSUser jest już w zasadzie ukończony (potem dodamy jeszcze jedną rzecz, ale to na koniec). Dlatego możemy wykonać kolejny commit do Git-a[iv].

Sprawdzamy, czy użytkownik jest zalogowany w Księstwie Sarmacji

Ponieważ integracja jest najważniejszą częścią naszej wtyczki, pierwsze co nas interesuje, to sprawdzenie czy użytkownik jest zalogowany w systemach Księstwa. Ponieważ, jeżeli tak jest, będziemy potrzebowali funkcji tyczących bieżącego użytkownika WordPress, musimy zrobić to w odpowiednim miejscu. Takim miejscem jest moment, gdy WP będzie miał już te funkcje przygotowane oraz będzie wiedział, kto jest aktywnym użytkownikiem. Wymaga to od nas spojrzenia na listę akcji i filtrów, w jakiej kolejności się wykonują[v] i gdzie możemy się „wpiąć".

Widzimy, że akcja „*set_current_user*", która ustawia użytkownika WP, występuje tuż przed akcją „*init*". Init jest jedną z częściej używanych akcji do „wpinania" się kodu wtyczek i szablonów, dzieje się wtedy, kiedy WordPress jest załadowany, ewentualny użytkownik rozpoznany i zalogowany (do WP) ale jeszcze nic nie zostało wypisane na wyjściu (przesłane do przeglądarki) czyli możemy np. bezpiecznie przeładować stronę. Wobec tego, „wpinamy" się w akcję:

```php
<?php
/**
 * Created by PhpStorm.
 * User: Łukasz Nowicki
 * Date: 18.02.2018
 * Time: 20:17
 */

namespace KS\Plugin\User;

/**
 * Class Plugin
 *
 * @package KS\Plugin\User
 */
class Plugin {

   /**
    * Plugin constructor.
    */
   function __construct() {
      add_action( 'init', [ $this, 'init' ] );
   }

   /**
    * Inicjalizacja WordPressa dobiegła końca
    */
   function init() {
      # ten kod wykona się, kiedy wszystko będzie załadowane
      # i gotowe do użycia ale jeszcze nic nie będzie wyświetlane
   }

}
```

Teraz możemy przystąpić do pisania kodu sprawdzającego, w metodzie „init”. Proszę zwrócić uwagę, że dzięki wykorzystaniu przestrzeni nazw i klas, nie musimy stosować funkcji w rodzaju:

```
function plugin_ks_integracja_action_init() {}
```

co bardzo ułatwia nam życie i sprawia, że kod jest czytelniejszy.

Kod sprawdzający zaczniemy od inicjalizacji klasy Integracja. Będziemy potrzebowali do tego danych konfiguracyjnych aplikacji. Z oczywistych względów nie przedstawię żadnych realnych danych, ale w repozytorium czytelnik znajdzie plik AppConfig-Sample.php z klasą AppConfig. Należy zmienić nazwę pliku (poprzez usunięcie -Sample) oraz uzupełnić dane autoryzacyjne według komentarzy. W tej chwili wygląda to tak:

```
function init() {
   $wp_user = wp_get_current_user();
   if ( 0 === $wp_user->ID ) {
      $this->require_authorization();
   }
}
```

Uważny czytelnik zauważy, że sprawdzamy czy użytkownik jest zalogowany w systemie WordPress. Jeżeli nie, to jest wywoływana metoda *require_authorization*. W tej metodzie, na początku, sprawdzimy czy przypadkiem użytkownik właśnie nie dokonał autoryzacji. Wygląda to tak:

```
function require_authorization() {
   $message = '';
   $this->integracja = new Integracja();
   $this->integracja->setConfiguration( ( new AppConfig() )->data() );
   $this->ks_user = $this->integracja->getUser();
   $this->ks_wynik = $this->integracja->getWynik();
   $error = (int)( isset( $this->ks_wynik['error'] ) ? $this->ks_wynik['error'] : -1 );
   if ( 666 === $error ) {
      $message = 'Dokonano zmiany w wymaganych uprawnieniach. Konieczna jest ' . $this->auth_link('ponowna autoryzacja') . '
w systemie Księstwa Sarmacji';
   } elseif ( 200 !== $error ) {
      $message = 'Wystąpił nieznany błąd. Konieczna jest ' . $this->auth_link('ponowna autoryzacja') . ' w systemie Księstwa
Sarmacji';
   }
   if ( '' != $message ) {
      $this->show_error( $message );
   }
}
```

Postępujemy tu praktycznie rzecz biorąc zgodnie z przykładem opisanym w pliku integracyjnym. Tyle, że ustawiamy sobie zmienną wskaźnik *error* w danych *Wynik* aby za każdym razem nie sprawdzać, czy jest ustawiona. Nawiasem mówiąc zalecałbym twórcom skryptu integracyjnego choćby taką poprawkę aby osoby korzystające były pewne, że dostaną informację. No i dobrze by było gdyby nie mieszano ustawicznie nazewnictwa polskiego i angielskiego bo jest to mylące, frustrujące a i prowadzi do podobnego działania, niestety.

Zauważymy tutaj również nową metodę *auth_link*. To takie ułatwienie, które zwraca nam łańcuch ze znacznikiem link (*<a>*) i źródłem do autoryzacji aplikacji. Wygląda tak:

```
function auth_link( $content ) {
   return '<a href="' . $this->integracja->loginURL() . '">' . $content . '</a>';
}
```

Poza tym występuje też metoda *show_error*, która wyświetla przekazany łańcuch i kończy pracę systemu. Później ją udoskonalimy, w tej chwili wygląda tak:

```
function show_error( $message ) {
   wp_die( $message );
}
```

Na koniec, dla jasności, należy wspomnieć o trzech zmiennych prywatnych w których przechowujemy odwołanie do klasy *Integracja*, oraz rezultat działania metod *getUser* i *getWynik* tej klasy. Cała nasza klasa wygląda na ten moment następująco:

```
<?php
/**
 * Created by PhpStorm.
 * User: Łukasz Nowicki
 * Date: 18.02.2018
 * Time: 20:17
 */

namespace KS\Plugin\User;

/**
 * Class Plugin
 *
 * @package KS\Plugin\User
 */
class Plugin {

   /**
    * @var Odwołanie do klasy Integracja
    */
   private $integracja;

   /**
    * @var Dane użytkownika KS
    */
   private $ks_user;
```

```
    /**
     * @var Wynik wywołania integracji KS
     */
    private $ks_wynik;

    /**
     * Plugin constructor.
     */
    function __construct() {
        add_action( 'init', [ $this, 'init' ] );
    }

    /**
     * Inicjalizacja WordPressa dobiegła końca
     */
    function init() {
        $wp_user = wp_get_current_user();
        if ( 0 === $wp_user->ID ) {
            $this->require_authorization();
        }
    }

    /**
     * @param $content Treść tego, co wyświetli się w linku
     *
     * @return string
     */
    function auth_link( $content ) {
        return '<a href="' . $this->integracja->loginURL() . '">' . $content . '</a>';
    }

    /**
     * @param $message
     */
    function show_error( $message ) {
        wp_die( $message );
    }

    /**
     * Autoryzacja właściwa
     */
    function require_authorization() {
        $message = '';
        $this->integracja = new Integracja();
        $this->integracja->setConfiguration( ( new AppConfig() )->data() );
        $this->ks_user = $this->integracja->getUser();
        $this->ks_wynik = $this->integracja->getWynik();
        $error = (int)( isset( $this->ks_wynik['error'] ) ? $this->ks_wynik['error'] : -1 );
        if ( 666 === $error ) {
            $message = 'Dokonano zmiany w wymaganych uprawnieniach. Konieczna jest ' . $this->auth_link('ponowna autoryzacja')
. ' w systemie Księstwa Sarmacji';
        } elseif ( 200 !== $error ) {
            $message = 'Wystąpił nieznany błąd. Konieczna jest ' . $this->auth_link('ponowna autoryzacja') . ' w systemie
Księstwa Sarmacji';
        }
        if ( '' != $message ) {
            $this->show_error( $message );
        }
    }

}
```

Możemy też zauważyć ten moment w naszym repozytorium[vi].

Użytkownik nie autoryzował aplikacji

Użytkownik nie autoryzował naszej aplikacji, wobec powyższego powinniśmy o taką poprosić. Najpierw jednak zorientować się, że autoryzacja nie nastąpiła. Sprawdzamy to poprzez testowanie zmiennej prywatnej *ks_user*. Jeżeli nie dokonano autoryzacji, zmienna przyjmie wartość **null**. W każdym razie to wykazał mi eksperyment. Jednak w swoim przykładzie NIA testuje tą zmienną nieco dokładniej i na inne przypadki, więc na wszelki wypadek je uwzględnię. Kod jest trywialny:

```
if ( is_null( $this->ks_user ) || empty( $this->ks_user ) || !isset( $this->ks_user ) ) {

    // nie dokonano autoryzacji
}
```

Tylko co wówczas możemy zrobić? O autoryzację poprosić, rzecz jasna. W tym celu po prostu uzupełniamy zawartość zmiennej *message*, dzięki której wiemy czy proces jest niewidzialny, czy też chcemy użytkownika o czymś poinformować:

```
$message = 'Aby przeglądać tę stronę musisz ' . $this->auth_link('autoryzować aplikację') . '.';
```

Potem następuje sprawdzenie czy zmienna *message* zawiera jakąś treść a jeżeli tak, to ta treść jest wyświetlana, to już było w poprzednim przykładzie.

Użytkownik autoryzował aplikację

Odnieśliśmy sukces, użytkownik chce z nami współpracować, autoryzował aplikację. Czyli zmienna prywatna *$ks_user* zawiera jakieś dane. Świetnie! Jaki powinien być kolejny krok? Musimy się zorientować, czy autoryzujący naszą aplikację użytkownik posiada już konto w naszym systemie, czy jeszcze nie. Musimy odwołać się do bazy danych WordPress. Zrobimy to za pomocą klasy *WP_User_Query*. Kod wygląda tak:

```
$user_query = new \WP_User_Query( array( 'meta_key' => 'KSI_paszport', 'meta_value' => $user['paszport'] ,  'fields' =>
'all'  ) );
```

Również dosyć trywialne. Sprawdzamy meta klucz użytkownika na obecność KSI_paszport (gdzie będziemy zapisywać ID paszportu) oraz na zgodność z przekazaną przez skrypt integracyjny wartością. Nie musimy się martwić o oczyszczenie łańcucha *$user['paszport']* ponieważ o to zadba klasa *WP_User_Query*.

Teraz pobieramy tablicę wyników. Jeżeli taki użytkownik istnieje, to wynikiem musi być jednoelementowa tablica. Gdy ją otrzymamy, możemy zalogować właściwego użytkownika a następnie przeładować stronę aby WordPress załadował sobie wszystkie odpowiednie rzeczy uznając użytkownika za pełnoprawnego uczestnika systemu. Zwróćcie uwagę, że WordPress sam nie zakończy wykonywania skryptu po użyciu funkcji przekierowującej, musimy o to zadbać sami. Razem wygląda to tak:

```
$user_query = new \WP_User_Query( array( 'meta_key' => 'KSI_paszport', 'meta_value' => $user['paszport'] ,  'fields' =>
'all'  ) );
$user_table = $user_query->get_results();
if ( is_array( $user_table ) && ( 1 === count( $user_table ) ) ) {
   $wp_user = $user_table[0];
   wp_clear_auth_cookie();
   wp_set_current_user ( $wp_user->ID );
   wp_set_auth_cookie  ( $wp_user->ID );
   $redirect_to = home_url();
   wp_safe_redirect( $redirect_to );
   exit();
}
```

Tutaj kończy się działanie tego fragmentu wtyczki, jeżeli mieszkaniec KS jest już sparowany z użytkownikiem WordPress. Jeżeli jednak nie, musimy to spowodować poprzez dodanie nowego użytkownika do CMS-a.

Dla zwiększenia bezpieczeństwa, procedura będzie przeprowadzona tylko wtedy, gdy skrypt integracyjny zwrócił nam pole *paszport*. Czego będziemy potrzebowali? Paszportu, który będzie wyróżnikiem, hasła (które nie będzie miało większego znaczenia – bo nie będziemy go używać w trakcie logowania się, autoryzacja nam wystarczy) oraz adresu email. Z adresem jest ten problem, że po dyskusji o RODO możemy przewidywać, że wkrótce będzie niedostępny. W związku z tym, jeżeli go nie będzie, to sobie go stworzymy.

Mając te dane, możemy stworzyć użytkownika i uzyskać nowy numer ID, robimy to tak:

```
if ( isset( $this->ks_user['paszport'] ) && ( '' !== $this->ks_user['paszport'] ) ) {
   $new_user_password = wp_generate_password(32,TRUE);
   $new_user_email = ( isset( $this->ks_user['email'] ) && ( '' != $this->ks_user['email'] ) ) ? $this->ks_user['email'] :
$this->ks_user['paszport'] . '@sarmacja.org';
   $new_user_id = wp_create_user( $this->ks_user['paszport'], $new_user_password, $new_user_email );
}
```

Teraz możemy dodać otrzymany nick (w tym przykładzie dodajemy też paszport) oraz uaktualnić informacje meta dzięki którym użytkownik będzie później rozpoznawalny przez naszą wtyczkę. Finalnie tworzymy obiekt użytkownika i przypisujemy mu rolę subskrybenta (jest to rola o najmniejszym znaczeniu w WordPress dzięki czemu zabezpieczamy się przed nieautoryzowaną modyfikacją systemu):

```
wp_update_user([
   'ID' => $new_user_id,
   'nickname' => $this->ks_user['nick'] . ' [' . $this->ks_user['paszport'] . ']'
]);
add_user_meta( $new_user_id, 'KSI_paszport', $this->ks_user['paszport'], TRUE );
add_user_meta( $new_user_id, 'KSI_data', $this->ks_user, TRUE );
$ready_user = new \WP_User( $new_user_id );
$ready_user->set_role('subscriber');
```

Użytkownik jest już gotowy. Pozostaje nam tylko zalogować użytkownika i przeładować system.

Pamiętacie, że kod do logowania już pisaliśmy? Teraz, ponieważ musimy użyć go drugi raz, przeniesiemy go do oddzielnej metody i sparametryzujemy:

```
function login_user_by_id( $user_id ) {
   wp_clear_auth_cookie();
   wp_set_current_user ( $user_id );
   wp_set_auth_cookie  ( $user_id );
   wp_safe_redirect( home_url() );
   exit();
}
```

Całość kodu pozwala nam już na pełną autoryzację użytkownika za pomocą wtyczki[vii].

Jak się wylogować?

Tutaj pojawia się problem. Jeżeli skorzystamy z mechanizmu WordPress do wylogowania, to pomimo tego, że zostaniemy poprawnie wylogowani jako użytkownik, to… nie zostaniemy wylogowani. Dlaczego? Ano, dlatego że pozostaje ciągle sesja autoryzacyjna, która grzecznie i skwapliwie nas zaloguje z powrotem. Co należy zrobić?

Rozwiązanie jest dosyć proste.

Musimy do konstruktora klasy *Plugin* dodać akcję *wp_logout*, która jest wywoływana właśnie w momencie wylogowywania:

```
add_action( 'wp_logout', [ $this, 'wp_logout' ] );
```

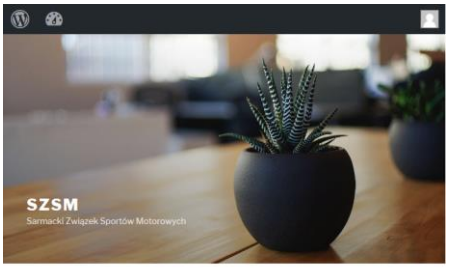
Teraz musimy zaprogramować samą metodę. Jest to trywialne i nieskomplikowane, polega – zgodnie z instrukcją autorów integracji – na usunięciu sesji.

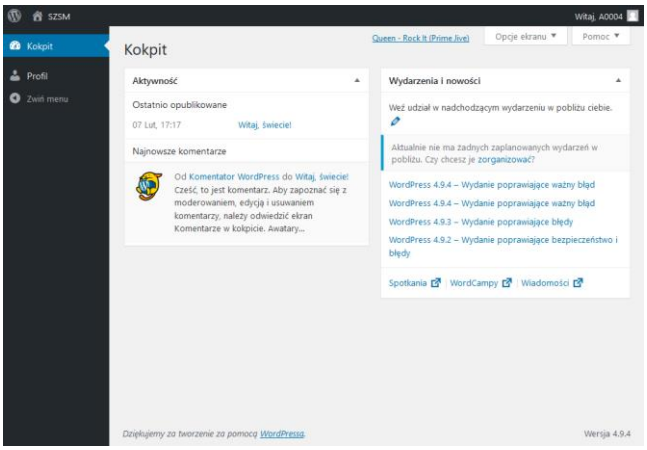
```
function wp_logout() {
   $_SESSION = [];
   if ( ini_get('session.use_cookies' ) ) {
      $params = session_get_cookie_params();
      setcookie( session_name(), '', time() - 36000,
         $params['path'], $params['domain'],
         $params['secure'], $params['httponly']
      );
   }
   session_destroy();
}
```

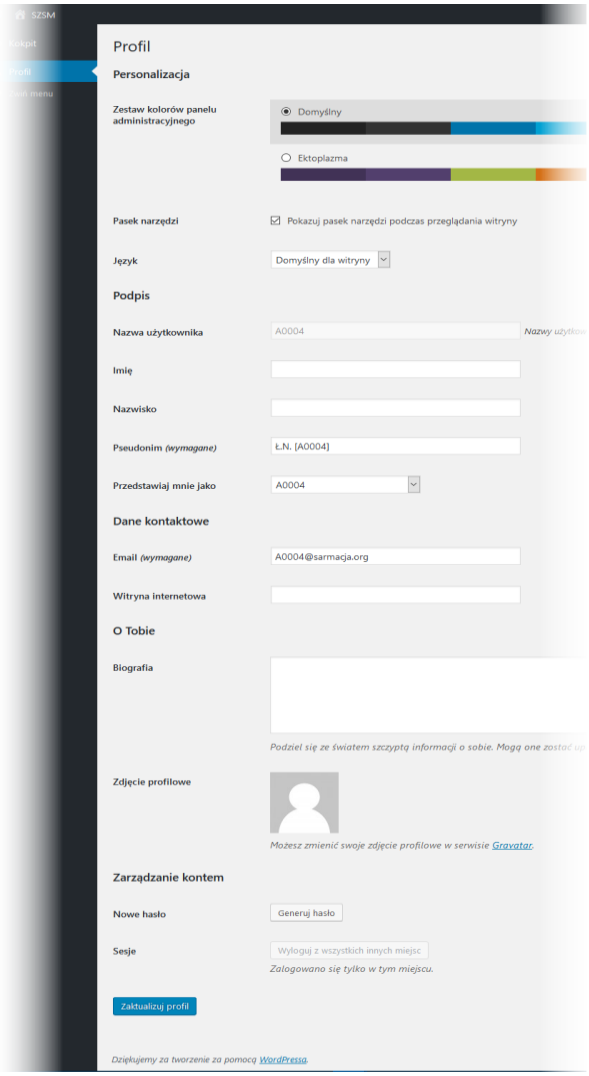
Robimy to w ten sposób, na początek czyszcząc sesję, potem ustawiając ją jako nieważną a na końcu ją niszcząc, co zapewnia nam wystarczający poziom bezpieczeństwa.

Wtyczka już gotowa, więc…

Nasza wtyczka już działa, jesteśmy prawidłowo logowani do systemu, możemy w nim „przebywać” oraz możemy się bezpiecznie wylogować. Schemat działania wygląda mniej więcej następująco:







Tryb wchodzenia na stronę nie jest skomplikowany. Czyli możemy uznać, że nasza wtyczka jest gotowa. Ale czy na pewno? Oczywiście spełnia swoje podstawowe wymagania, ale zawsze chcielibyśmy czegoś więcej. Dla przykład, przejdźmy do profilu użytkownika (obrazek po lewej stronie). Czego nam tutaj brakuje?

Ewidentnie informacji z systemów informatycznych Księstwa Sarmacji!

Skąd zdobyć te informacje? Ano, z naszych meta danych przypisanych użytkownikowi. Jak je wyświetlić? O tym właśnie sobie teraz napiszemy.

Do tego rodzaju operacji mamy dwie bardzo przydatne akcje - *show_user_profile* oraz *edit_user_profile*. Jak łatwo się domyślić, jedna jest wywoływana w momencie wyświetlania profilu użytkownika a druga – w momencie wyświetlania ekranu edycji użytkownika. Wykorzystamy tą pierwszą, jako że i tak nie możemy edytować danych użytkownika Księstwa.

Do tego celu stworzymy sobie kolejną klasę, którą nazwiemy *Profil*. Klasa będzie miała zaledwie dwie metody. Jedną magiczną, czyli konstruktor – to tam właśnie będziemy przypisywać akcję:

```
function __construct() {
   add_action( 'show_user_profile', [ $this, 'show_user_profile' ] );
}
```

Oraz *show_user_profile*, czyli metodę pokazującą zawartość. Na potrzeby niniejszej publikacji ograniczymy się do trzech parametrów, ale jak widać tworzony HTML może być nieograniczony:

```
function show_user_profile( $user ) {
   $meta = get_user_meta( $user->ID, 'KSI_data', TRUE );
   ?>
   <h2>Dane profilowe Księstwa Sarmacji</h2>
   <table class="form-table">
      <tbody>
      <tr class="user-ks-paszport">
         <th>Numer paszportu</th>
         <td><?php echo $meta['paszport']; ?></td>
      </tr>
      <tr class="user-ks-nick">
         <th>Nick</th>
         <td><?php echo $meta['nick']; ?></td>
      </tr>
      <tr class="user-ks-bank">
         <th>Na koncie:</th>
         <td><?php echo number_format( $meta['money'], 2, ',', ' ' ); ?> lt</td>
      </tr>
      </tbody>
   </table>
   <?php
}
```

Koniec

Nasza wtyczka jest gotowa.

Możesz pobrać ją tutaj:

https://github.com/lukasznowicki/sarmacja-integracja

Powodzenia w rozwijaniu swoich zainteresowań!

Kod źródłowy:

AppConfig-Sample

```php
<?php
/**
 * Created by PhpStorm.
 * User: Łukasz Nowicki
 * Date: 18.02.2018
 * Time: 21:40
 */

namespace KS\Plugin\User;


class AppConfig {

    /**
     * @var string ID aplikacji, np. S00001
     */
    public $id = '';

    /**
     * @var string Nazwa aplikacji, np. "Moja super aplikacja!"
     */
    public $nazwa = '';

    /**
     * @var string Tajny klucz aplikacji, do wygenerowania w panelu
     */
    public $klucz = '';

    /**
     * @var string Adres zwrotny, gdzie aplikacja ma się przeładować po
     *      autoryzacji, np. http://app.sarmacja.org/
     */
    public $adres_powrotu = '';

    /**
     * @var bool Czy z poziomu aplikacji mają być dostępne przelewy?
     */
    public $przelewy = TRUE;

    /**
     * @var int Limit pojedynczego przelewu, w libertach
     */
    public $przelew_jednorazowy = 10;

    /**
     * @var int Limit przelewów na dobę
     */
    public $przelew_doba = 100;

    /**
     * @var bool Czy używamy powiadomień?
     */
    public $powiadomienia = TRUE;

    public function data() {
        return [
            'appId'     => $this->id,
            'appName'   => $this->nazwa,
            'appSecret' => $this->klucz,
            'adress'    => $this->adres_powrotu,
            'options'   => [
                'przelew'             => $this->przelewy,
                'jednorazowyPrzelew' => $this->przelew_jednorazowy,
                'dniowyPrzelew'      => $this->przelew_doba,
                'powiadomienie'      => $this->powiadomienia,
            ],
        ];
    }

}
```

Autoloader

```php
<?php

namespace KS\Plugin\User;

/**
 * Class Autoloader
 *
 * @package KS\Plugin\User
 */
class Autoloader {

   /**
    * @var array
    */
   protected $namespaces = [];

   /**
    * @return bool
    */
   public function register() {
      spl_autoload_register( [ $this, 'load_class' ] );

      return TRUE;
   }

   /**
    * @param $prefix
    * @param $dir
    * @param bool $prepend
    *
    * @return bool
    */
   public function add_namespace( $prefix, $dir, $prepend = FALSE ) {
      $prefix = trim( $prefix, '\\' ) . '\\';
      $dir    = rtrim( $dir, DIRECTORY_SEPARATOR ) . '/';
      if ( isset( $this->namespaces[ $prefix ] ) === FALSE ) {
         $this->namespaces[ $prefix ] = [];
      }
      if ( $prepend ) {
         array_unshift( $this->namespaces[ $prefix ], $dir );
      } else {
         array_push( $this->namespaces[ $prefix ], $dir );
      }

      return TRUE;
   }

   /**
    * @param $class
    *
    * @return bool|string
    */
   public function load_class( $class ) {
      $prefix = $class;
      while ( FALSE !== $pos = strrpos( $prefix, '\\' ) ) {
         $prefix         = substr( $class, 0, $pos + 1 );
         $relative_class = substr( $class, $pos + 1 );
         $mapped_file    = $this->load_mapped_file( $prefix, $relative_class );
         if ( $mapped_file ) {
            return $mapped_file;
         }
         $prefix = rtrim( $prefix, '\\' );
      }

      return FALSE;
   }

   /**
    * @param $prefix
    * @param $relative_class
    *
    * @return bool|string
    */
   protected function load_mapped_file( $prefix, $relative_class ) {
      if ( isset( $this->namespaces[ $prefix ] ) === FALSE ) {
         return FALSE;
      }
      foreach ( $this->namespaces[ $prefix ] as $base_dir ) {
         $file = $base_dir . str_replace( '\\', '/', $relative_class ) . '.php';
         if ( $this->file_call( $file ) ) {
```

```
            return $file;
         }
      }

      return FALSE;
   }

   /**
    * @param $file
    *
    * @return bool
    */
   protected function file_call( $file ) {
      if ( is_readable( $file ) ) {
         require_once $file;

         return TRUE;
      }

      return FALSE;
   }

}
```

Plugin

```php
<?php
/**
 * Created by PhpStorm.
 * User: Łukasz Nowicki
 * Date: 18.02.2018
 * Time: 20:17
 */

namespace KS\Plugin\User;

/**
 * Class Plugin
 *
 * @package KS\Plugin\User
 */
class Plugin {

   /**
    * @var Odwołanie do klasy Integracja
    */
   private $integracja;

   /**
    * @var
    */
   private $user_profile;

   /**
    * @var Dane użytkownika KS
    */
   private $ks_user;

   /**
    * @var Wynik wywołania integracji KS
    */
   private $ks_wynik;

   /**
    * Plugin constructor.
    */
   function __construct() {
      add_action( 'init', [ $this, 'init' ] );
      add_action( 'wp_logout', [ $this, 'wp_logout' ] );
      if ( is_admin() ) {
         $this->user_profile = new Profile();
      }
   }

   /**
    * Inicjalizacja WordPressa dobiegła końca
    */
   function init() {
      $wp_user = wp_get_current_user();
      if ( 0 === $wp_user->ID ) {
         $this->require_authorization();
      }
   }

   /**
    * Autoryzacja właściwa
    */
   function require_authorization() {
      $message          = '';
      $this->integracja = new Integracja();
      $this->integracja->setConfiguration( ( new AppConfig() )->data() );
      $this->ks_user  = $this->integracja->getUser();
      $this->ks_wynik = $this->integracja->getWynik();
      $error          = (int) ( isset( $this->ks_wynik['error'] ) ? $this->ks_wynik['error'] : - 1 );
      if ( isset( $this->ks_user ) && isset( $this->ks_user['paszport'] ) && ( - 1 === $error ) ) {
         $error = 200;
      }
      if ( 666 === $error ) {
         $message = 'Dokonano zmiany w wymaganych uprawnieniach. Konieczna jest ' . $this->auth_link( 'ponowna autoryzacja'
) . ' w systemie Księstwa Sarmacji';
      } elseif ( 200 !== $error ) {
         $message = 'Wystąpił nieznany błąd. Konieczna jest ' . $this->auth_link( 'ponowna autoryzacja' ) . ' w systemie
Księstwa Sarmacji';
      }
      if ( is_null( $this->ks_user ) || empty( $this->ks_user ) || ! isset( $this->ks_user ) ) {
         $message = 'Aby przeglądać tę stronę musisz ' . $this->auth_link( 'autoryzować aplikację' ) . '.';
```

```
        }
        if ( '' != $message ) {
            $this->show_error( $message );
        }
        $user_query = new \WP_User_Query( [
            'meta_key'   => 'KSI_paszport',
            'meta_value' => $this->ks_user['paszport'],
            'fields'     => 'all',
        ] );
        $user_table = $user_query->get_results();
        if ( is_array( $user_table ) && ( 1 === count( $user_table ) ) ) {
            $wp_user = $user_table[0];
            update_user_meta( $wp_user->ID, 'KSI_data', $this->ks_user );
            $this->login_user_by_id( $wp_user->ID );
        }
        if ( isset( $this->ks_user['paszport'] ) && ( '' !== $this->ks_user['paszport'] ) ) {
            $new_user_password = wp_generate_password( 32, TRUE );
            $new_user_email    = ( isset( $this->ks_user['email'] ) && ( '' != $this->ks_user['email'] ) ) ? $this-
>ks_user['email'] : $this->ks_user['paszport'] . '@sarmacja.org';
            $new_user_id       = wp_create_user( $this->ks_user['paszport'], $new_user_password, $new_user_email );
            wp_update_user( [
                'ID'       => $new_user_id,
                'nickname' => $this->ks_user['nick'] . ' [' . $this->ks_user['paszport'] . ']',
            ] );
            add_user_meta( $new_user_id, 'KSI_paszport', $this->ks_user['paszport'], TRUE );
            add_user_meta( $new_user_id, 'KSI_data', $this->ks_user, TRUE );
            $ready_user = new \WP_User( $new_user_id );
            $ready_user->set_role( 'subscriber' );
            $this->login_user_by_id( $new_user_id );
        }
        exit( 'Coś poszło bardzo nie tak.' );
    }

    /**
     * @param $content Treść tego, co wyświetli się w linku
     *
     * @return string
     */
    function auth_link( $content ) {
        return '<a href="' . $this->integracja->loginURL() . '">' . $content . '</a>';
    }

    /**
     * @param $message
     */
    function show_error( $message ) {
        wp_die( $message );
    }

    /**
     * @param $user_id
     */
    function login_user_by_id( $user_id ) {
        wp_clear_auth_cookie();
        wp_set_current_user( $user_id );
        wp_set_auth_cookie( $user_id );
        wp_safe_redirect( home_url() );
        exit();
    }

    /**
     * Wylogowanie
     */
    function wp_logout() {
        $_SESSION = [];
        if ( ini_get( 'session.use_cookies' ) ) {
            $params = session_get_cookie_params();
            setcookie( session_name(), '', time() - 36000,
                $params['path'], $params['domain'],
                $params['secure'], $params['httponly']
            );
        }
        session_destroy();
    }

}
```

Profile

```php
<?php
/**
 * Created by PhpStorm.
 * User: Łukasz Nowicki
 * Date: 25.02.2018
 * Time: 19:07
 */

namespace KS\Plugin\User;

/**
 * Class Profile
 *
 * @package KS\Plugin\User
 */
class Profile {

   /**
    * Profile constructor.
    */
   function __construct() {
      add_action( 'show_user_profile', [ $this, 'show_user_profile' ] );
   }

   /**
    * @param $user
    */
   function show_user_profile( $user ) {
      $meta = get_user_meta( $user->ID, 'KSI_data', TRUE );
      ?>
      <h2>Dane profilowe Księstwa Sarmacji</h2>
      <table class="form-table">
         <tbody>
         <tr class="user-ks-paszport">
            <th>Numer paszportu</th>
            <td><?php echo $meta['paszport']; ?></td>
         </tr>
         <tr class="user-ks-nick">
            <th>Nick</th>
            <td><?php echo $meta['nick']; ?></td>
         </tr>
         <tr class="user-ks-bank">
            <th>Na koncie:</th>
            <td><?php echo number_format( $meta['money'], 2, ',', ' ' ); ?>
               lt
            </td>
         </tr>
         </tbody>
      </table>
      <?php
   }

}
```

Spis treści

[i] PSR-0: https://www.php-fig.org/psr/psr-0/, PSR-4: https://www.php-fig.org/psr/psr-4/
[ii] https://github.com/lukasznowicki/sarmacja-integracja/commit/588584c977e0c8026dc89586cfb64e8a0e8afeb4
[iii] http://fc.sarmacja.org/viewtopic.php?f=387&t=12815#p128970
[iv] https://github.com/lukasznowicki/sarmacja-integracja/commit/f3326308adcdb9f33cfbb5e3c55c9d4ae1df77aa
[v] http://rachievee.com/the-wordpress-hooks-firing-sequence/
[vi] https://github.com/lukasznowicki/sarmacja-integracja/commit/59badcf6a3fd6bf969f3f83bc7affc6813642e18
[vii] https://github.com/lukasznowicki/sarmacja-integracja/commit/74f60495620e2803657e8f0ba2a924cf179e60a9