Davelopweb - Rapport de stage

Semaine 8/8

Sommaire :

1 - Introduction :

*Davelopweb est une entreprise qui accompagne **la transformation numérique** des indépendants, associations et TPE. Elle propose 3 principaux services :*

- ***Création de site web** : Avec WordPress l'entreprise propose de créer des sites adaptés au entreprises, tout en optimisant au maximum celui-ci pour être au courant des habitudes des utilisateurs, que l'on puisse ajouter des fonctionnalités sans code, et avoir un bon référencement naturel.*

- ***Développement CRM** : En tant que CEO de l'entreprise, Dave a développé pour lui un outil permettant de remplir son emploi du temps, faire une liste de tâches, enregistrer ses contacts, etc. Une application de gestion d'activité entièrement adaptée à ses besoins nommée Yucca.*

  Ensuite, en ayant parlé de son application par hasard à deux de ses clients, il a remarqué qu'ils seraient intéressés par cette application. Pour chaque client intéressé, il note leurs besoins et leur fait une application Yucca totalement personnalisée.

- ***Installation Cloud privée** : L'entreprise propose également de la sauvegarde sur un cloud grâce à l'application NextCloud. Ça permet de sauvegarder automatiquement votre site grâce à une copie sur NextCloud et sur votre PC/MAC mensuellement pour plus de sécurité.*

*Davelopweb est une **micro-entreprise** composé d'une seule personne, Dave Delalleau, un **développeur web full-stack**. Elle se situe au 161 avenue de l'Hippodrome, 33200 Eysines et on peut contacter son CEO à contact@davelopweb.fr.*

2 –Mes missions :

1– Mise en conformité d'un plug-in WordPress :

***Contexte** : En tant que CEO de l'entreprise, Dave à codé un **plug-in** en PHP permettant de **sauvegarder automatiquement un site WordPress** (BDD, code, fonctionnalités…) vers un cloud tous les mois. Cependant après avoir envoyé un mail auprès du staff de WordPress, il s'avère que son code n'est pas conforme et ne peut donc pas être validé.*

***Ma mission** : En tant que stagiaire dans l'entreprise, mon rôle est de **modifier le code** du plugin pour être conforme et pouvoir être valider par WordPress.*

*Pour ce faire je dispose du **mail envoyé par le staff** de WordPress qui indique les points sur lesquels le code n'est pas correct, avec les lignes de code concerné et des informations complémentaires pour m'aider à régler les problèmes.*

Il y a 5 points à régler, voici en un pour vous montrer la mise en forme du mail :

Incorrect Stable Tag

In your readme, your 'Stable Tag' does not match the Plugin Version as indicated in your main plugin file.

Save to Nextcloud/readme.txt:6:Stable tag: 1.0.0
Save to Nextcloud/STN.php:7: * * Version: 3.0

Your Stable Tag is meant to be the stable version of your plugin, not of WordPress. For your plugin to be properly downloaded from WordPress.org, those **values** need to be the same. If they're out of sync, your users won't get the right version of your code.

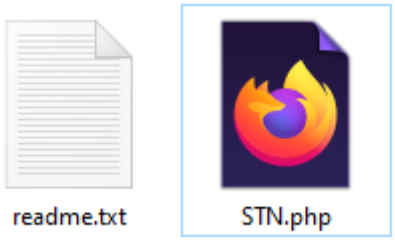
We recommend you use Semantic Versioning (aka SemVer) for managing versions:

- https://en.wikipedia.org/wiki/Software_versioning
- https://semver.org/

Please note: While *currently* using the stable tag of trunk currently works in the Plugin Directory, it's not actually a supported or recommended method to indicate new versions and has been known to cause issues with automatic updates.

We ask you please properly use tags and increment them when you release new versions of your plugin, just like you update the plugin version in the main file. Having them match is the best way to be fully forward supporting.

Le dossier du plugin contient deux fichiers :



*STN.php (Save To NextCloud) est le fichier principal contenant le **code php** du plugin alors que l'autre contient seulement **quelques informations** telles que la version du plug-in, une aide pour l'installer, etc.*

*Le premier problème cité dans le mail consister seulement à **changer la ligne Version** dans le fichier readme.txt car elle ne correspondait pas à celle indiqué dans le fichier STN.php. Dave l'ayant changé avant que j'arrive, je n'en parlerai pas ici.*

A– Changer les noms de class/fonction :

Voici la problématique :

Generic function/class/define/namespace names

All plugins must have unique function names, namespaces, defines, and class names. This prevents your plugin from conflicting with other plugins or themes. We need you to update your plugin to use more unique and distinct names.

*Synthétiquement, tous les plug-ins de WordPress sont **stockés au même endroit**. Le problème et que si deux class ou fonction ont le **même nom** cela peut engendrer des problèmes alors au vu des noms un peu simpliste choisi, je dois les changer.*

*Dans la suite du mail, il y a écrit les class et méthodes posant un problème. Le plug-in s'appelant « Save To NextCloud », j'ai **renommé les class et méthodes indiquées** en leur mettant « **stn_** » devant leur nom.*

B– Utiliser l'API WordPress plutôt que des cURL :

Rentrons dans le vif du sujet. Voici la problématique :

Using CURL Instead of HTTP API

WordPress comes with an extensive HTTP API that should be used instead of creating your own curl calls. It's both faster and more extensive. It'll fall back to curl if it has to, but it'll use a lot of WordPress' native functionality first.

https://developer.wordpress.org/plugins/http-api/

Please note: If you're using CURL in 3rd party vendor libraries, that's permitted. It's in your own code unique to this plugin (or any dedicated WordPress libraries) that we need it corrected.

CURL est une interface en ligne de commande, destinée ***à récupérer le contenu d'une ressource accessible par un réseau*** *informatique. Dave avait utilisé un cURL pour récupérer et envoyer les fichiers WordPress nécessaires à la sauvegarde.*

Voici le code :

```
    //envoi sur Nextcloud
    //procédure curl
    $ch = curl_init();
    curl_setopt($ch, CURLOPT_URL,
get_option('url_dlwcloud').'/remote.php
/webdav'.get_option('folder_dlwcloud').basename($file_cloud));
    curl_setopt($ch, CURLOPT_USERPWD,
get_option('login_dlwcloud').":".get_option('pass_dlwcloud'));
    curl_setopt($ch, CURLOPT_PUT, 1);
    $fh_res = fopen($file_cloud, 'r');
    curl_setopt($ch, CURLOPT_INFILE, $fh_res);
    curl_setopt($ch, CURLOPT_INFILESIZE, filesize($file_cloud));
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
    curl_setopt($ch, CURLOPT_BINARYTRANSFER, TRUE); // --data-binary
    $response = curl_exec($ch);
    $error = curl_error($ch);
```

Ps : je n'explique pas le code j'ai dû l'effacer complètement pour le refaire et donc je n'ai jamais eu à le comprendre.

*Cependant, le staff de WordPress nous a dit qu'il fallait utiliser leur **API** plutôt qu'un cURL si c'est possible comme dans notre cas. Il a donc fallu que je **modifie l'intégralité de la fonction qui permettait la récupération et l'envoie des fichiers** WordPress nécessaires à la sauvegarde.*

*Cette **mission a été la principale que j'ai effectuée** pendant le stage. Voici comment j'ai procédé :*

*Pour commencer, j'étais totalement perdu et inondé d'informations. Un long code déjà fait, que des méthodes inconnu pour moi. C'est donc là qu'à commencer un long **moment de recherche et d'analyse de code**. Mon objectif était tout d'abord de comprendre le fonctionnement et les méthodes principales du code. Après cette partie analyse de code à suivi une partie recherche sur l'API de WordPress. Ce qui en ressort, c'est une méthode qui va m'être très utile par la suite : **wp_remote_request()***

```
wp_remote_request( string $url, array $args = array() ): array|WP_Error
```

Elle prend en paramètre l'url à qui effectuer l'action (GET, POST, PUT…) et un tableau contenant l'en-tête et le corps de la requête.

*À la suite de ces recherches, j'ai dû réfléchir à comment j'allais pouvoir récupérer et sauvegarder les fichiers nécessaires à la sauvegarde d'un site WordPress, en outre, à **l'algorithme**. Et voici ce qu'il en est : Je dois **compresser les fichiers** dans un fichier .tar que je vais **enregistrer dans le serveur** de Dave. Puis **envoyer celui-ci au lien NextCloud** enregistrer grâce à un input.*

Voici le code permettant de créer et enregistrer le fichier .tar dans le serveur :

```php
/**

    Ce fichier permet de créer un tar dans le FTP contenant tout ce dont on a besoin pour sauvegarder WordPress

**/

// Fichier de la BDD
$config_file = ABSPATH . "wp-config.php";
$htaccess_file = ABSPATH . ".htaccess";
$content_file = ABSPATH . "wp-content/";
$okFile = ABSPATH . date('Ymd'). "tarOK.txt";

// Programme dans 60 secondes checkok
wp_schedule_single_event(time()+60, 'checkok');

// Ajoute le script de la BDD dans le fichier
shell_exec("mysqldump -h ".DB_HOST." -u ".DB_USER." --password='".DB_PASSWORD."' ".DB_NAME." > ".stn_save_to_nextcloud::getFileBDD());

// Créer le tar sur le FTP et créer un fichier lorsque le tar est terminé
exec("tar -czf ".stn_save_to_nextcloud::getFileCloud()." ".stn_save_to_nextcloud::getFileBDD()." ".$config_file." ".$htaccess_file." ".
$content_file." --exclude='*/cache' || touch " .$okFile);
```

*On peut y voir les chemins des fichiers nécessaires pour la sauvegarde d'un site WordPress, puis, deux commande linux. La première permet de **récupérer le script de la BDD** du site WordPress et le second permet de **mettre tous les fichiers dans un fichier .tar qui va s'enregistrer dans le serveur**.*

*Puis, j'ai codé **l'envoi du fichier** vers NextCloud grâce à la méthode wp_remote_request() en **POST**. Je ne peux malheureusement pas vous montrer le code ici, mais nous le verrons un peu plus tard.*

*Une fois **l'algorithme fonctionnel** sur mon ordinateur, je **présentai mon code** à Dave qui **l'essaya sur un client**. Il fut très surpris avant d'essayer en voyant que le site du client faisait 10 Giga. On était persuadé à la suite de nombreux essaye et vérification de mon code sur mon ordinateur que cela aller marcher, mais malheureusement ça ne marchait pas et on comprit immédiatement pourquoi. **La requête POST ne supporte pas des fichiers aussi lourds que 10 Giga**. En effet sur mon ordinateur, je n'avais essayé qu'avec un petit fichier pour ne pas perdre de temps. J'ai donc dû **revoir entièrement l'algorithme** pour pouvoir faire l'envoi d'un fichier lourd. Et ceci signa la fin de ma première semaine de stage.*

C– Utilisation des chunks :

*La deuxième semaine commença et comme dit précédemment mon objectif principal a été de **trouver un moyen de faire passer un lourd fichier** dans une requête POST, mais cela s'est annoncé plus dur que prévue. En effet, ça **n'est pas possible** de faire passer aussi gros avec l'API de WordPress. Il a donc fallu trouver un autre moyen. Après quelques recherches, j'ai trouvé une idée assez intéressante dont le principe était **de diviser le gros fichier en petits morceaux** appelé des « Chunks », les envoyer une par une grâce à la requête POST dans un dossier temporaire dans NextCloud puis donner l'ordre de regrouper ces morceaux.*

Voici le code pour diviser en chunks le fichier compressé :

```php
// Taille maximum des bout de fichier à envoyer
$MAX_SIZE = 50 * 1048576;
```

```php
// Ouvre le fichier en lecture
$handle = fopen(stn_save_to_nextcloud::getFileCloud($date), 'rb');
```

```php
// Tant que le fichier n'est pas entierrement lu
while(!feof($handle)){
```

```php
$chunkNum += 1;

// Prépare le headers
$headers = array(
  'content-type'  => 'application/binary',
  // Identifiant et mdp de NextCloud
  'Authorization' => 'Basic ' . base64_encode(get_option("login_dlwcloud") . ":" . get_option("pass_dlwcloud")),
);

// Prépare les arguments
$args = array(
  // Requête de création
  'method' => 'PUT',
  'timeout' => 30,
  'redirection' => 5,
  'httpversion' => '1.0',
  'blocking' => true,
  'headers' => $headers,
  // Lit le contenu du fichier tar jusqu'à (si possible) $MAX_SIZE bits
  'body' => fread($handle, $MAX_SIZE),
  'cookies' => array()
);

// Envoi la requête (créer le morceau chunk dans le dossier uuid)
$resSendChunk = wp_remote_request(get_option('url_dlwcloud').'/remote.php/dav/uploads/' . get_option('login_dlwcloud').
'/' . stn_save_to_nextcloud::getCurrentFile("uuid") ."/chunk".$chunkNum, $args);
```

L'algorithme se base sur les méthodes php fopen() et fread(). On commence par calculer la taille maximum d'un chunk (50Mo ici). Ensuite, ***on ouvre le fichier compressé en lecture*** *grâce à fopen(fichier, lecture) qu'on met dans la variable $handle. Puis, tant que le fichier n'a pas été lu entièrement (feof() signifie File End Of File),* ***on envoie un morceau du fichier*** *grâce à la méthode fread() qui prend en argument le fichier à lire puis la taille à laquelle il doit s'arrêter de lire. À chaque envoi d'un morceau, $handle s'incrémente automatiquement de $MAX_SIZE car le fichier est ouvert en lecture et fread() lit le fichier donc à chaque tour de boucle fread() reprend là où il s'est arrêté la dernière fois. Le tout jusqu'à finir le fichier complet.*

Ces morceaux vont être placés dans un dossier créé pour les accueillir dont le nom est composé d'un « ***uuid*** *». Un uudi est un identifiant créer à l'aide d'une méthode qui* ***garantit l'unicité*** *de celui-ci. Cela permet dans notre cas que le dossier créé n'en* ***écrase pas un autre****.*

Voici le code qui regroupe les morceaux de chunks :

```php
// Prépare le headers
$headers = array(
  'content-type'  => 'application/binary',
  // Login et mot de passe rentrée dans les champs
  'Authorization' => 'Basic ' . base64_encode(get_option("login_dlwcloud") . ":" . get_option("pass_dlwcloud")),
  // Va être envoyer vers la destination
  'Destination'   => get_option('url_dlwcloud').'/remote.php/dav/files/'.get_option('login_dlwcloud').get_option('folder_dlwcloud') .
  date('Ymd')."_wp_save_".stn_save_to_nextcloud::getDomain().".tar",
);

// Préapre les arguments
$args = array(
  // Requête qui change d'emplacement
  'method' => 'MOVE',
  'timeout' => 30,
  'redirection' => 5,
  'httpversion' => '1.0',
  'blocking' => true,
  'headers' => $headers,
  'body' => array(),
  'cookies' => array()
);

// Envoi la requête (Rassemble les chunks dans 'Destination')
$resMergeChunk = wp_remote_request(get_option('url_dlwcloud').'/remote.php/dav/uploads/' . get_option('login_dlwcloud').
'/' . stn_save_to_nextcloud::getCurrentFile("uuid") . "/.file", $args );
```

*On y retrouve encore une fois une requête HTTP mais cette fois-ci avec la méthode **MOVE**. Celle-ci permet de **changer d'emplacement une entité**. Dans notre cas on effectue l'action sur notre dossier de chunks suivie de "/.file" pour que ça **prenne tous les fichiers à l'intérieur du dossier**. Puis on les emmène vers l'attribut Destination qui n'est autre que le fichier .tar final dans NextCloud qui va accueillir la sauvegarde. Cette requête permet donc de **reconstituer le fichier**.*

*Cette fois-ci, en étant sûr de mon coup, on essaya une deuxième fois sur le client qui avait 10Giga à sauvegarder. Mais malheureusement, encore une fois, **ça n'a pas marché**. Il y avait un problème **au niveau de l'envoi des chunks, tous ne s'envoyais pas**. Le fait d'envoyer trop de requêtes stoppai l'algorithme. On n'a pas su exactement pourquoi, mais cela peut être dû à deux choses. Soit l'envoie trop nombreux de requêtes bloque l'envoie à partir d'un certain stade, soit le code est trop long à s'exécuter alors il s'arrête. En effet pour envoyer 10Giga de donnée avec des chunks de 50Mo ça fait 200 requêtes à envoyer.*

D– Suivie de l’envoie des chunks grâce à une BDD:

En bref, il a fallu encore une fois ***changer la structure du code*** *pour qu’il puisse fonctionner, et cette fois, on n’y est pas allé de main morte. Tout d’abord, j’ai commencé par rajouter un* ***système de base de données*** *pour* ***savoir lorsque les chunks ont arrêté de s’envoyer****.*

Voici la table qui va me servir :

```php
// Récupère la class wpdb grâce à une variable global
global $wpdb;

// Va chercher la type de la base
$charset_collate = $wpdb->get_charset_collate();

// Nom de la base
$nameTable=$wpdb->prefix.'stn_dir_chunk';
// Requête la la création de la table
$sql="CREATE TABLE IF NOT EXISTS $nameTable (
            id_dir_chunk int(11) NOT NULL auto_increment,
            uuid varchar(50) DEFAULT NULL,
            nbChunk int(11) DEFAULT NULL,
            uploadTime int(20) DEFAULT NULL,
            finish boolean DEFAULT false,
            PRIMARY KEY (id_dir_chunk)
        )$charset_collate;";

// Va chercher le doc pour modifier la BDD
require_once(ABSPATH.'wp-admin/includes/upgrade.php');
// Applique la requête
dbDelta($sql);
```

La table est créée ***automatiquement lors de l’activation du plug-in*** *dans WordPress et supprimer grâce à un drop table lors de la désactivation de celui-ci.*

Description des champs :

- *Uuid : Nom du dossier temporaire*
- *nbChunk : Nombre de chunk dont l’envoi est un succès*
- *uploadTime : enregistre la méthode time() au moment où l’envoi du chunk est un succès*
- *Finish : Indique si tous les chunks ont bien été envoyé*

*Je rappelle que l'objectif du plug-in est de **sauvegarder AUTOMATIQUEMENT** un site WordPress et tous ses fichiers importants à une fréquence choisie. Cela signifie que **le code doit fonctionner en parfaite autonomie** une fois lancer. Notre problème est que **le code s'arrête** lors de l'envoi d'un nombre trop important de chunk. Il a fallu donc créer une méthode **qui vérifie si l'envoie des chunks s'est arrêté** et qui dans ce cas, relance la fonction. Et qui, bien évidemment, lorsqu'on la relance, reprenne là où elle s'était arrêtée au lieu de tout recommencer.*

C'est pourquoi j'ai créé la méthode suivante :

```php
// Vérifie si les chunks ont planté et relance le si necessaire
1 reference | 0 overrides
function cron_gendarme($nbChunkMax){
    // Si tout les chunks n'ont pas été envoyés
    if($nbChunkMax != intval(stn_save_to_nextcloud::getCurrentFile("nbChunk"))){
        // Si ça fait longtemps qu'un chunk n'a pas été envoyés
        if(time() > stn_save_to_nextcloud::getCurrentFile("uploadTime") + 50){
            // Ranlance la fonction
            stn_save_to_nextcloud::save_in_nextCloud();
        }
        else{
            // Programme dans 30s le gendarme
            wp_schedule_single_event(time()+30, 'gendarme', array($nbChunkMax));
        }
    }
}
```

*Son fonctionnement est simple. Elle prend **en paramètre le nombre de chunks total qu'il est attendu pour le fichier** (200 pour celui de 10Giga) puis**, vérifie si est atteint** grâce à la fonction getCurrentFile qui prend en paramètre un champ et qui va chercher dans la BDD les informations sur le fichier en cours de sauvegarde. Si tout les chunks ne sont pas encore envoyés, **on vérifie depuis combien de temps de dernier chunk reçu a été envoyé**. Sachant qu'en moyenne un chunk met 5 secondes à s'envoyer et en prenant large, j'ai dit que si on **n'a pas reçu de chunk depuis 50 secondes**, c'est que l'algorithme s'est arrêté alors il faut le **relancer**. Sinon, c'est que l'envoi de chunk est toujours en cours alors je demande à cron_gendarme() de se relancer dans 30 secondes.*

*Bien évidemment, pour que ça fonctionne, il faut aussi **adapter la partie d'envoi des chunks pour enregistrer les valeurs actuelles dans la BDD**, mais aussi que lorsqu'on relance le script, **il reprenne là où il s'était arrêté**. Et c'est ce que je m'apprêtais à faire avant que l'heure ne sonne la fin de ma deuxième semaine de stage.*

Ma troisième semaine de stage débuta et je repris là où je m'étais arrêté. Je dus donc maintenant, ***adapter la partie d'envoi des chunks pour enregistrer les valeurs actuelles dans la BDD****, mais aussi que lorsqu'on relance le script,* ***il reprenne là où il s'était arrêté****.*

Voici les modifications apportées :

```php
// Récupère la class wpdb grâce à une variable global
global $wpdb;

// Taille maximum des bout de fichier à envoyer
$MAX_SIZE = 50 * 1048576;
// Récupère le nombre de chunk déja télécharger
$chunkNum = intval(stn_save_to_nextcloud::getCurrentFile("nbChunk"));
// Ouvre le fichier en lecture
$handle = fopen(stn_save_to_nextcloud::getFileCloud($date), 'rb');

// Calcule ce qu'on a déjà lu
$alreadyRead = $chunkNum * $MAX_SIZE;
// Si on a déja lu quelques choses
if($alreadyRead > 0){
    // fread sur ce qu'on a déjà lu pour pouvoir reprendre à partir de là
    fread($handle, $alreadyRead);
}


// Récupère le nombre de chunk que devra avoir le fichier
$nbChunkMax = ceil(filesize(stn_save_to_nextcloud::getFileCloud($date)) / $MAX_SIZE);

// Active le gendarme qui vérifie si la boucle ne s'arrete pas
wp_schedule_single_event(time()+80, 'gendarme', array($nbChunkMax));

// Tant que le fichier n'est pas entierrement lu
while(!feof($handle)){
    // Si la connexion avec NextCloud est correct
    if(stn_save_to_nextcloud::is_NextCloud_good()){

        $chunkNum += 1;

        // Prépare le headers
        $headers = array(
          'content-type'  => 'application/binary',
          // Identifiant et mdp de NextCloud
          'Authorization' => 'Basic ' . base64_encode(get_option("login_dlwcloud") . ":" . get_option("pass_dlwcloud")),
        );
```

```php
// Prépare les arguments
$args = array(
  // Requête de création
  'method' => 'PUT',
  'timeout' => 30,
  'redirection' => 5,
  'httpversion' => '1.0',
  'blocking' => true,
  'headers' => $headers,
  // Lit le contenu du fichier tar jusqu'à (si possible) $MAX_SIZE bits
  'body' => fread($handle, $MAX_SIZE),
  'cookies' => array()
);

// Envoi la requête (créer le morceau chunk dans le dossier uuid)
$resSendChunk = wp_remote_request(get_option('url_dlwcloud').'/remote.php/dav/uploads/' . get_option('login_dlwcloud').
'/' . stn_save_to_nextcloud::getCurrentFile("uuid") ."/chunk".$chunkNum, $args);

// Nouvelles données
$data = array("nbChunk"  => $chunkNum,
              "uploadTime"=>time());

// Lorsque
$where = array("finish" => false);

// Execute la requête
$wpdb->update($wpdb->prefix.'stn_dir_chunk', $data, $where);

// Si c'est le dernier chunk
if($chunkNum == $nbChunkMax){
    // Si le gandarme est programmer
    if (wp_next_scheduled ( 'gendarme', array($nbChunkMax))) {
        // On le stop
        wp_clear_scheduled_hook('gendarme', array($nbChunkMax));
    }
}
```

*Au début de l'algorithme, on **regarde dans la BDD combien de chunks ont déjà était envoyé**, on calcule donc le nombre de bit déjà lu du fichier, puis **on reprend l'algo à partir de ce niveau**. En relançant l'algo, on reprendra donc là où on s'était arrêté. De plus, à la fin de chaque envoi, **on incrémente de 1 le champ "nbChunk" de la BDD** et met à jour le champ "uploadTime" pour permettre à l'algorithme de savoir où l'on en est. Enfin, lorsque tous les chunks sont envoyé, il faut arrêter le cron_gendarme().*

*Une fois cet algo fonctionnel, **Dave envoya de nouveau le code au staff de WordPress** et en attendant la réponse, il me demanda de **rajouter plein de petit modification/fonctionnalités** qui vont rendre plus simple et pratique l'utilisation du plug-in.*

*Tout d'abord, rajouter un **système d'envoi de mail**. Le principe est simple, en cas d'erreur de connexion avec NextCloud ou lorsque tout s'est bien déroulé, il faut **envoyer un mail à l'utilisateur**.*

E– Tester la connexion avec NextCloud:

Pour tester la connexion avec NextCloud, j'envoie une requête simple et regarde le retour :

```php
// Prépare le headers
$headers = array(
    // Login et mot de passe rentrée dans les champs
    'Authorization' => 'Basic ' . base64_encode(get_option("login_dlwcloud") . ":" . get_option("pass_dlwcloud")),
);

// Lance la requête de test de connexion
$nextcloud_response = wp_remote_head(get_option('url_dlwcloud').'/remote.php/dav/files', array('headers' => $headers));
```

*Une requête de **méthode HEAD**. C'est une méthode qui permet de **tester la connexion**. Ça marche comme une requête GET, mais sans retourner ce que l'on get. Ce qui importe ici, c'est simplement **le retour de la réponse**. Si la réponse est correcte alors **la connexion avec NextCloud est bonne**.*

*J'ai rajouté cette fonction de test de connexion avec NextCooud **au début de chaque requête vers NextCloud** comme ceci :*

```php
// Si la connexion avec NextCloud est correct
if(stn_save_to_nextcloud::is_NextCloud_good()){
    // Faire la requete
}
// Si la connexion avec NextCloud n'est pas correct
else{
    // Stop la sauvegarde, supprime toutes les choses gardées inutilement et envoie un mail pour prévenir de l'erreur
    stn_save_to_nextcloud::there_is_an_error();
}
```

Si la connexion n'est pas bonne, j'appelle la fonction there_is_an_error() que voici :

```php
// Fonction qui supprime toute les choses inutile à stocker et arrête la sauvegarde lors d'une erreur
4 references | 0 overrides
static function there_is_an_error(){

    // Supprime l'occurence de la BDD
    include("Controleurs/DeleteChunksBdd.php");

    // J'envoie un mail pour prévenir de l'erreur
    stn_save_to_nextcloud::sendMail(true);

    // Stop la sauvegarde
    exit;

}
```

*Celle-ci permet de **supprimer de la BDD l'occurrence qui gère la sauvegarde**, puis d'envoyer un mail grâce à la méthode sendMail(). Il est à noter que j'aurais aussi dû **supprimer les chunks déjà enregistrer**, mais sachant qu'ils sont dans NextCloud et que s'il y une erreur c'est que je ne peux pas me connecter à NextCloud, **je ne peux pas le faire**. Alors j'ai **rajouté cette fonctionnalité en début de code**, lorsque l'on*

F - Gérer les erreurs:

lance une sauvegarde ça va supprimer automatiquement les déchets des sauvegardes précédentes.

Voici la méthode sendMail :

```php
// Si il y a une erreur
if($error){

    // Prépare l'objet du mail
    $type = "ERROR";
    // Prépare le message du mail
    $texte_mail="Erreur d'envoi de la sauvegarde de votre site sur votre espace Nextcloud. Veuillez impérativement vérifier les informations données (URL, identifiant, mot de passe, dossier de sauvegarde et email). N'oubliez pas de mettre un slash en début et en fin du champ.";

}
// Si il n'y a pas d'erreur
else{

    // Prépare l'objet du mail
    $type = "OK";
    // Prépare le message du mail
    $texte_mail="La sauvegarde de votre site est terminée, veuillez vérifier dans votre dossier Nextcloud.";

}

// Objets
$sujet = 'Sauvegarde du site sur le domaine '. stn_save_to_nextcloud::getDomain() .' > ' . $type;
$headers[] = 'From: Save to Nextcloud <savetonextcloud@'. stn_save_to_nextcloud::getDomain().'>';
$headers[] = 'Content-Type: text/html; charset=UTF-8';

// Envoi du mail
wp_mail(get_option('email_dlwcloud'), $sujet, $texte_mail, $headers);
```

Rien de plus simple à comprendre. Wp_mail() prend en argument une adresse mail, un message ext… Et ***envoie le mail à l'adresse correspondante****.*

Voici le bout de code qui permet de ***supprimer les fichiers créés dans le FTP pour la sauvegarde*** *:*

```php
// Liste de tout les fichiers restant possible dans le FTP
$filesInFtp = glob(ABSPATH . "wp_save_*");

// Pour tous les fichiers restant
foreach($filesInFtp as $file){
    // Le supprime
    unlink($file);
}
```

La méthode glob() permet de ***récupérer tous les fichiers qui commencent par wp_save****. Puis je boucle sur tous les fichiers et* ***les supprime*** *grâce à unlink().*

Pour finir, je dus aussi rajouter une dernière fonctionnalité. Pour faire une sauvegarde, l'utilisateur doit indiquer le chemin dans NextCloud avec lequel il veut que la

*sauvegarde soit rangée. Dave voulait que **si l'on indique un chemin et que certains dossiers du chemin n'existent pas. On les crées**.*

```php
// Enlève les slash en début et fin
$userDestination = substr(get_option('folder_dlwcloud'), 1, strlen(get_option('folder_dlwcloud')) - 2);
// Récupère tout les dossier du chemin dans un array
$tab_dir = explode('/', $userDestination);
// Prépare la variable à incrémenter pour chaque dossier
$chemin = '';

// Pour chaque dossier du chemin
foreach ($tab_dir as $dir){

    // Ajoute le dossier au chemin. Cette variable permet de vérifier pour chaque dossier un par un avec son chemin
    $chemin .= '/' . $dir;

    // Prépare les arguments de la requête
    $args = array(
      // Ordre de récupérer
      'method' => 'GET',
      'timeout' => 30,
      'redirection' => 5,
      'httpversion' => '1.0',
      'blocking' => true,
      'headers' => array(
        // Identifiant et mdp de NextCloud
        'Authorization' => 'Basic ' . base64_encode(get_option("login_dlwcloud") . ":" . get_option("pass_dlwcloud")),
      ),
      'body' => array(),
      'cookies' => array()
    );

    // Envoi de la requête
    $resGetUserDestination = wp_remote_request(get_option('url_dlwcloud').'/remote.php/dav/' .
    'files/'.get_option('login_dlwcloud').$chemin, $args );
```

*Pour pouvoir tester tous les dossiers un par un. Je dois **récupérer le nom de tous les dossiers**. Tout d'abord, j'utilise la méthode substr() qui permet de récupérer une partie d'un string pour enlever les slashes en début et fin de chemin. Ensuite, j'utilise la méthode explode qui permet **de récupérer un array de chaîne de caractère en indiquant un séparateur**. Ici, le slash. Ensuite, pour chacun des dossiers, je fais une requête GET.*

*Je sais que le code réponse d'une requête sur un **dossier qui n'existe pas est 404**, alors **je teste la réponse de la requête GET** comme cela :*

```php
// Si le chemin de l'utilisateur n'existe pas
if($resGetUserDestination["response"]["code"] == 404){

    // Prépare les arguments de la requête
    $args = array(
      // Ordre de créer un dossier
      'method' => 'MKCOL',
      'timeout' => 30,
      'redirection' => 5,
      'httpversion' => '1.0',
      'blocking' => true,
      'headers' => array(
        // Identifiant et mdp de NextCloud
        'Authorization' => 'Basic ' . base64_encode(get_option("login_dlwcloud") . ":" .
        get_option("pass_dlwcloud")),
       ),
      'body' => array(),
      'cookies' => array()
    );

    // Envoi de la requête
    $resCreateDestination = wp_remote_request(get_option('url_dlwcloud').'/remote.php/dav/files/'.
    get_option('login_dlwcloud').$chemin, $args );

}
```

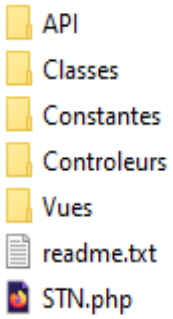
Si le dossier n'existe pas, je le crée *avec une* ***requête MKCOL*** *qui indique la création d'un dossier. Avec en argument le chemin que j'incrémente pour chaque dossier.* ***Une fois le foreach terminé, tous les dossiers du chemin sont créés****. C'est ainsi que s'acheva ma troisième semaine de stage en espérant avoir la réponse du staff de WordPress en début de semaine suivante.*

3 – Rapport finale de stage :

L'intitulé de ma mission de stage était de **modifier le code** d'un plugin WordPress pour être conforme et pouvoir être validé. Pour ce faire je disposais du **mail envoyé par le staff** de WordPress qui indiquais les points sur lesquels le code n'était pas correct, avec les lignes de code concernées et des informations complémentaires pour m'aider à régler les problèmes.

Cependant, depuis ce jour les choses ont évoluer et à la suite de la validation de mon code par l'équipe de WordPress j'ai dû améliorer le plugin en lui ajoutant des fonctionnalités.

Voici en cette fin de stage à quoi ressemble l'arborescence de mon code :



Sachant que mon tuteur de stage va probablement par la suite devoir reprendre mon code j'ai essayé de faire les choses proprement.

Le fichier readme.txt est obligatoire pour un plugin WordPress est contient les récapitulatif de toute les mises à jour et les versions du plugin qui ont était créés.

En voici une partie, il est donc écrit ce que j'ai fait en fin de stage :

```
= 1.0.4 =

Don't use the ok files anymore. Resume zip possible

= 1.0.5 =

Added choice for frequency, day and time of backup. Possibility to notify several
people by email.

= 1.0.6 =

Cannot run two backups at the same time. Change the functionality of buttons.

= 2.0.0 =

Add 2 functions for the API : List of the user's parameters, list of his backups.
Enable auto update of themes.
Possibility to activate automatic updates only during a period of 48 hours, starting two hours after the backup.
Possibility to block automatic updates. The plugin becomes compatible with WordPress multisite.

= 3.0.0 =

Addition of a new menu, thus modification of the visual of the menus.
Possibility to restore a backup.
Reorganization of the structure (directory/includes).
```

Plongeons-nous plus profondément dans le code pour voir comment fonctionne t'il :

```php
// ---------- Les classes ---------- //


// Classe qui permet de sauvegarder un site WordPress
include("Classes/stn_save_to_nextcloud.php");

// Classz qui permet de restaurer un site WordPress
include("Classes/stn_restore_from_nextcloud.php");


// ---------- L'API ---------- //


// Crée une route dans l'API qui permet de récupérer la liste des paramètres de l'utilisateur en format json
include("API/ListUserParam.php");

// Crée une route dans l'API qui permet de récupérer le nombre de sauvegarde et pour chacune, sa date et son état.
include("API/ListSave.php");

// ---------- Autres fonctionnalités ---------- //


// Permet gérer les sauvegarde automatiques
include("Controleurs/ManageAutoSave.php");

// Permet créer les crons
include("Controleurs/CreateCron.php");

// Permet d'afficher le bouton d'auto update pour les themes
include("Controleurs/ShowAutoUpdateTheme.php");

// Permet gérer l'affichage des menus
include("Controleurs/ManageMenu.php");

// Permet créer les paramètres d'utilisateur
include("Controleurs/CreateUserParam.php");


// ---------- Les vues ---------- //


// Fonction qui permet d'afficher le visuel de la page de sauvegarde
include("Vues/vue_save.php");

//  Fonction qui permet d'afficher le visuel de la page de restauration
include("Vues/vue_restore.php");
```

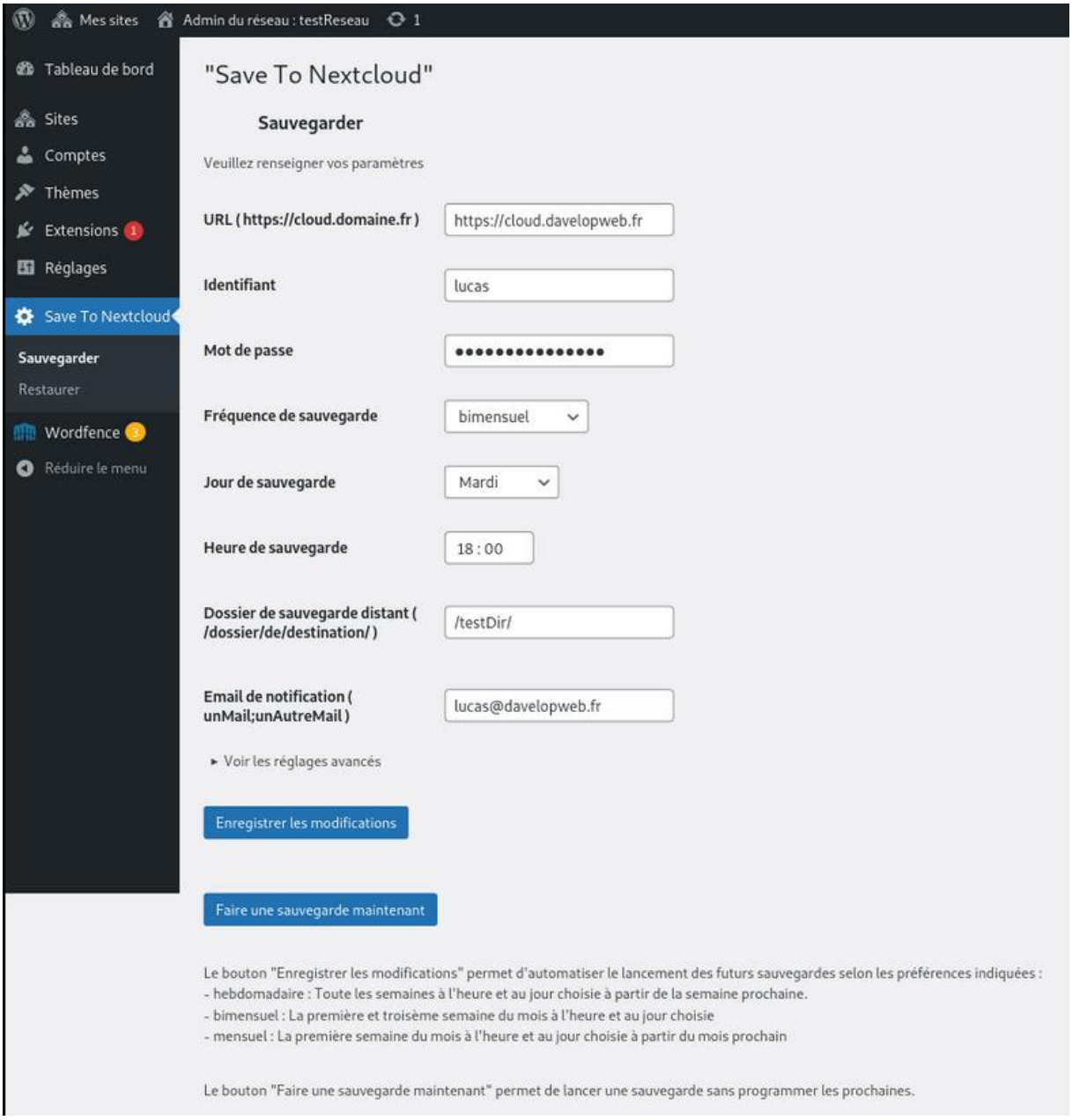
Ci-dessus se présente le fichier principal du plugin, STN.php, dans celui-ci j'y est mis tous les includes nécessaire au bon fonctionnement du code. Dans ce code on peut y voir les deux fonctionnalités principales du code que j'ai fait.

- La sauvegarde
- La restauration

Chacune de ces fonctionnalité dispose de son menu dans le plugin ainsi que de sa classe et de sa vue.

A– La sauvegarde :

Voici le visuel de la page :



On peut y voir les différentes options programmées. Chaque option est un paramètre stocké pour chaque utilisateur qui est enregistrer grâce à cette petite partie de code :

```php
/*menu administration*/
if ( is_admin() ){

    // Fonction qui enregistres les paramètres d'utilisateurs
    0 references
    function stn_savetonextcloud_settings() { // whitelist options
      register_setting( 'nextcloud-group', 'url_dlwcloud' );
      register_setting( 'nextcloud-group', 'login_dlwcloud' );
      register_setting( 'nextcloud-group', 'pass_dlwcloud' );
      register_setting( 'nextcloud-group', 'frequency_dlwcloud');
      register_setting( 'nextcloud-group', 'day_dlwcloud' );
      register_setting( 'nextcloud-group', 'hour_dlwcloud' );
      register_setting( 'nextcloud-group', 'folder_dlwcloud' );
      register_setting( 'nextcloud-group', 'email_dlwcloud' );
      register_setting( 'nextcloud-group', 'nb_save_dlwcloud' );
      register_setting( 'nextcloud-group', 'auto_update_dlwcloud' );

    }

    // Enregistre les settings
    add_action( 'admin_init', 'stn_savetonextcloud_settings' );

}
```

Add_action permet que la méthode se déclenche lors de l'initialisation d'un écran d'administration ou d'un script. Lors que clique sur le bouton on nous envoie vers la page option.php qui lance cette méthode et qui enregistre les paramètres de l'utilisateur. De plus, on lance la méthode qui permet de calculer la prochaine date de sauvaugarde à effectuer selon les paramètres de l'utilisateurs et on programme une sauvegarde à cette date.

Il y a aussi le bouton « Sauvegarder maintenant » qui déclenche une sauvegarde sans programmer les suivantes. Voici comment fonctionne une sauvegarde.

Tout d'abord la méthode save_in_ftp est lancé. Son fonctionnement est très bien expliqué dans les commentaires :

```php
// Fonction qui télécharge les fichiers WP dans le serveur FTP et supprime les traces de l'ancienne sauvegarde s'il y en a
1 reference | 0 overrides
static function save_in_ftp($relance=false, $date=null, $time=null){

    // Si aucune sauvegarde n'est en cour, initialise la sauvegarde
    if(stn_save_to_nextcloud::backup_in_progress("stn_file_in_zip") == 0){

        // Supprime les chunks sauvegardés et gardés dans NextCloud
        include(PLUGIN_PATH . "Controleurs/DeleteChunksNextCloud.php");

        // Supprime l'occurence de la BDD
        include(PLUGIN_PATH . "Controleurs/DeleteChunksBdd.php");

        // Supprime s'il y en a les ancien fichiers sauvegardés et gardés dans le FTP
        include(PLUGIN_PATH . "Controleurs/DeleteFileFtp.php");

        // Ajoute une valeur dans la BDD
        include(PLUGIN_PATH . "Controleurs/AddDataFileInZip.php");
    }

    // Créer et envoi au ftp le fichier zip contenant tout ce qu'il faut pour sauvegarder WordPress
    include(PLUGIN_PATH . "Controleurs/CreateZip.php");

}
```

En résumant cette méthode commence par initialiser la sauvegarde en plusieurs étapes :

- Supprime les fichiers, dossier, occurrence dans la BDD indésirable
- Ajoute une occurrence dans la BDD pour la sauvegarde qui va démarrer

Puis, la sauvegarde démarre. Elle met dans un zip tous les fichiers et dossier nécessaire à la sauvegarde. Les dossiers sont gérés grâce à un système de récurrence. Parfois, le site est tellement lourd que la code s'arrête, il y a donc un système de reprise du code si jamais il s'arrête qui est mis en place automatiquement.

A titre d'exemple de code, voici comment le zip est créer et comment y sont ajouter les fichiers (pas les dossier) :

```php
// Liste des fichiers à mettre dans le zip
$filesInZip = array(
    "wp-config.php" => ABSPATH . "wp-config.php",
    ".htaccess" => ABSPATH . ".htaccess",
    "wp_save_bdd_" . $date ."_".$time."_".stn_save_to_nextcloud::getDomain().".sql" =>  stn_save_to_nextcloud::getFileBDD($date, $time),
);

// Instancie le zip
$zip = new ZipArchive();
// Créer un nouveau zip à la route param 1
$zip->open(stn_save_to_nextcloud::getFileCloud($date, $time), ZipArchive::CREATE);

// Pour tout les fichiers du tableau
foreach($filesInZip as $name => $file){
    // Ajoute le fichier
    $zip->addFile($file, $name);
}
```

Après avoir créer le zip il faut le transféré à NextCloud, c'est alors que la deuxième fonction ce lance :

```php
// Fonction qui transfert le zip au NextCloud indiqué
2 references | 0 overrides
static function save_in_nextCloud($date, $time){

    // Variable qui permet de gérer les erreurs
    $error = false;

    // Si aucune sauvegarde n'est en cour
    if(stn_save_to_nextcloud::backup_in_progress("stn_dir_chunk") == 0){

        // Créer le dossier temporaire SaveChunk qui contenir tout les morceaux du zip
        include(PLUGIN_PATH . "Controleurs/CreateFolder.php");

        // Ici je créer l'occurence initial
        include(PLUGIN_PATH . "Controleurs/AddDataChunk.php");

    }

    // Envoi les morceaux de fichier à SaveChunk
    include(PLUGIN_PATH . "Controleurs/SendChunk.php");

    // Créer le dossier inquiqué par l'utilisateur dans NextCLoud s'il n'existe pas
    include(PLUGIN_PATH . "Controleurs/CheckForDestination.php");

    // Fusionne tout les fichier dans un zip
    include(PLUGIN_PATH . "Controleurs/MergeChunk.php");

    // Passe la valeur de "finish" de la db à true
    include(PLUGIN_PATH . "Controleurs/FinishDbTrue.php");

    // Supprime les anciennes sauvegardes s'il y en a trop
    stn_save_to_nextcloud::deleteSaves($date, $time);

    // Envoie du mail de reponse correct
    stn_save_to_nextcloud::sendMail(false);

    // Supprime les fichiers du FTP
    include(PLUGIN_PATH . "Controleurs/DeleteFileFtp.php");


}
```

En résumant cette méthode commence par initialiser le transfert en plusieurs étapes :

- Crée un dossier invisible dans NextCloud (dans un dossier spécial)
- Ajoute une occurrence dans la BDD pour le transfert qui va démarrer

Puis, le transfert démarre. Le fichiers zip est d'abord diviser en plusieurs morceaux appelé chunk (Ce processus est nécessaire car on ne peut pas envoyer de gros fichiers avec des requête http). Les chunks sont envoyé un par un à NextCloud dans le dossier invisible. On fait ensuite, une requête MOVE qui regroupe tous les morceaux à l'endroit où l'utilisateur à décider. Puis, selon le nombre de sauvegarde l'utilisateur à choisie de garder on en supprime si nécessaire. On finit ensuite par lui envoyer un mail et supprimer le zip du serveur.

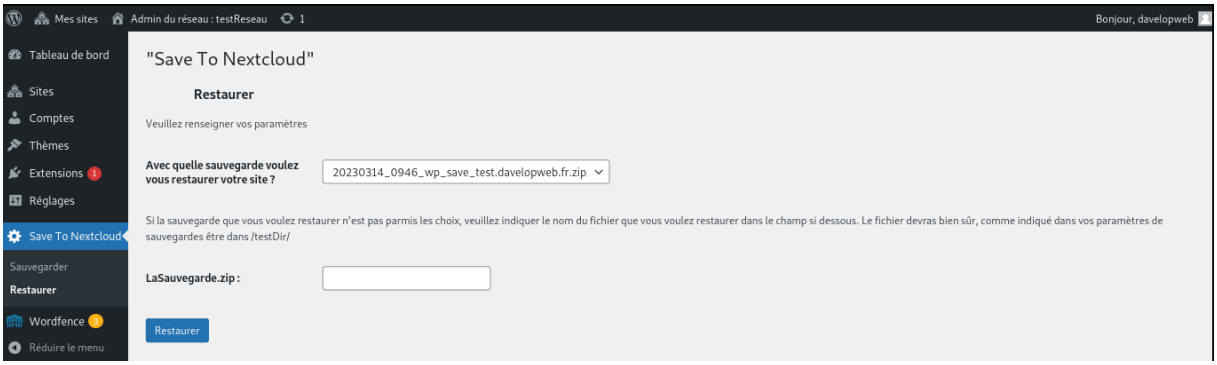
A titre d'exemple de code, voici comment les chunks sont regroupés :

```php
// Si la connexion avec NextCloud est correct
if(stn_save_to_nextcloud::is_NextCloud_good()){

    // Prépare le headers
    $headers = array(
        'content-type'  => 'application/binary',
        // Login et mot de passe rentrée dans les champs
        'Authorization' => 'Basic ' . base64_encode(get_option("login_dlwcloud") . ":" . get_option("pass_dlwcloud")),
        // Va être envoyer vers la destination
        'Destination'   => stn_save_to_nextcloud::getFileNextCloud($date, $time),
    );

    // Préapre les arguments
    $args = array(
      // Requête qui change d'emplacement
      'method' => 'MOVE',
      'timeout' => 30,
      'redirection' => 5,
      'httpversion' => '1.0',
      'blocking' => true,
      'headers' => $headers,
      'body' => array(),
      'cookies' => array()
    );

    // Envoi la requête (Rassemble les chunks dans 'Destination')
    $resMergeChunk = wp_remote_request(get_option('url_dlwcloud').'/remote.php/dav/uploads/' . get_option('login_dlwcloud')
    . '/' . stn_save_to_nextcloud::getCurrentFile("uuid") . "/.file", $args );


}
// Si la connexion avec NextCLoud est incorrect
else{
    // Stop la sauvegarde, supprime toutes les choses gardées inutilement et envoie un mail pour prévenir de l'erreur
    stn_save_to_nextcloud::there_is_an_error();
}
```

Dans le header ont met l'attributs Destination qui va accueillir les chunk, ont utilise la méthode MOVE qui sert à changer l'emplacement et on met dans fait la requête sur tous les fichiers qui sont dans le dossier des chunks.

B– La restauration :

Voici le visuel de la page :



Dans cet partie la page est beaucoup plus simple, sans option. Il faut juste choisir une des options ou alors marqué le nom du fichier de votre sauvegarde et la restauration se lance.

La restauration s'exécute grâce à deux fonctions que voici :

```php
// Permet d'executer la restauration côté FTP
0 references | 0 overrides
static function restore($saveFile){

    // Si le fichier old n'a pas fini d'être supprimer
    if (is_dir(ABSPATH . "old-wp-content")){
        // On le supprime
        stn_restore_from_nextcloud::rrmdir(ABSPATH . "old-wp-content");
    }

    // Permet de récupérer la sauvegarde dans le FTP
    include(PLUGIN_PATH . "Controleurs/getSave.php");

    // Permet d'extraire le contenue du zip dans le dossier FichiersRestauration
    include(PLUGIN_PATH . "Controleurs/ExtractionZip.php");

    // Permet de déplacer les fichiers pour effecutuer la restauration
    include(PLUGIN_PATH . "Controleurs/MoveFiles.php");

}
```

```php
// Restaure la BDD et nettoie le FTP
0 references | 0 overrides
static function restore_BDD($extractDir){

    // Execute le script de la BDD pour la restaurer
    include(PLUGIN_PATH . "Controleurs/RestoreBDD.php");

    // Supprime tout les fichiers/dossiers créés pour faire la restauration
    include(PLUGIN_PATH . "Controleurs/CleanFTP.php");

}
```

Pour commencer on importe la sauvegarde sur le serveur et on extrait son contenue. Ensuite, on déplace le contenue pour remplacer les fichiers et dossier actuel par les anciens. Puis on lance le script de la BDD qu'on avait précédemment créer lors de la sauvegarde. Et pour finir on nettoie tout les fichiers et dossiers créer ou stockée pour faire la restauration.

Il est à noté qu'une option est également disponible qui permet de garder le nom de domaine actuel si la sauvegarde à était faite depuis un autres site. Ce qui permet de

faire des changements de nom de domaine facilement en restaurant l'ancien site vers le nouveau nom de domaine.

Voici la partie du code qui sert à déplacer le dossier wp-content qui contient tout le contenue du site :

```php
// Tant que le code précédent n'est pas terminé on reste dans la boucle
while(true){
    // Si le code précédent est terminé
    if(!is_dir(ABSPATH . "old-wp-content/plugins/stn")){
        // Lance dans 10s la restauration de la BDD
        wp_schedule_single_event(time()+1,'restore_BDD_cron', array($extractDir));
        // On sors de la boucle
        break;
    }
}

// Déplace le dossier wp-content
rename($extractDir . "/wp-content", ABSPATH . "wp-content");
```

Lorsque l'on déplace ce dossier, le code du plugin est supprimé pour être remplacer alors le code s'arrête. Donc je fais une boucle permet de programmer la suite dans 1 seconde se qui laisse le temps au dossier wp-content d'être remplacé et donc le code reprendra avec les nouveau fichier et dossier du plugin. Il est à noté que j'ai fait en sorte que la version du plugin ne soit pas changée lors d'une restauration car ça pourrait poser des problèmes.

C– Conclusion:

Au cours de mon stage, j'ai pu mettre en pratique mes compétences en développement telles que PHP et MySQL au service de l'entreprise. J'ai pu gagner en expérience et connaissance du développement web en touchant à un des sites les plus utilisé dans ce domaine de nos jours, WordPress.

Je suis fier du travail accompli et je suis convaincu que le plugin de sauvegarde et de restauration que j'ai développé sera très utile pour les clients de Davelopweb. Je suis reconnaissant envers mon tuteur de Davelopweb pour cette opportunité de stage enrichissante et je suis impatient de poursuivre ma carrière dans le développement informatique.

Rapport de stage – semaine 8
Lucas BOUTEVIN SANCÉ
SIO2B