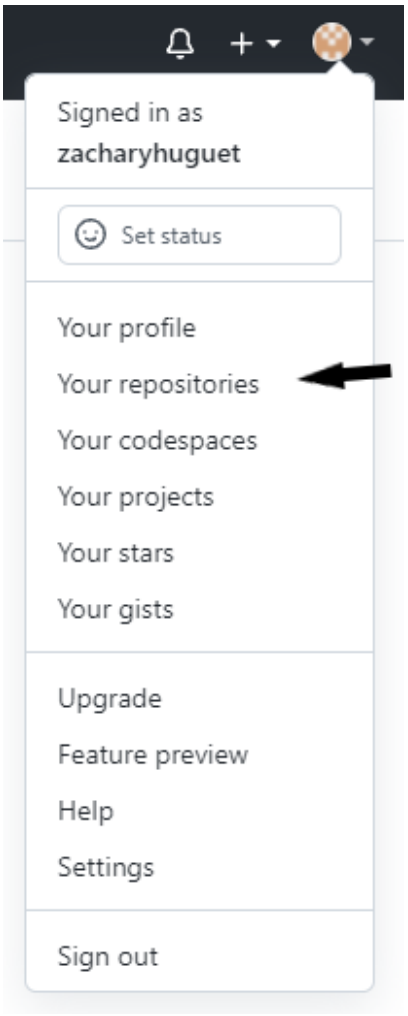
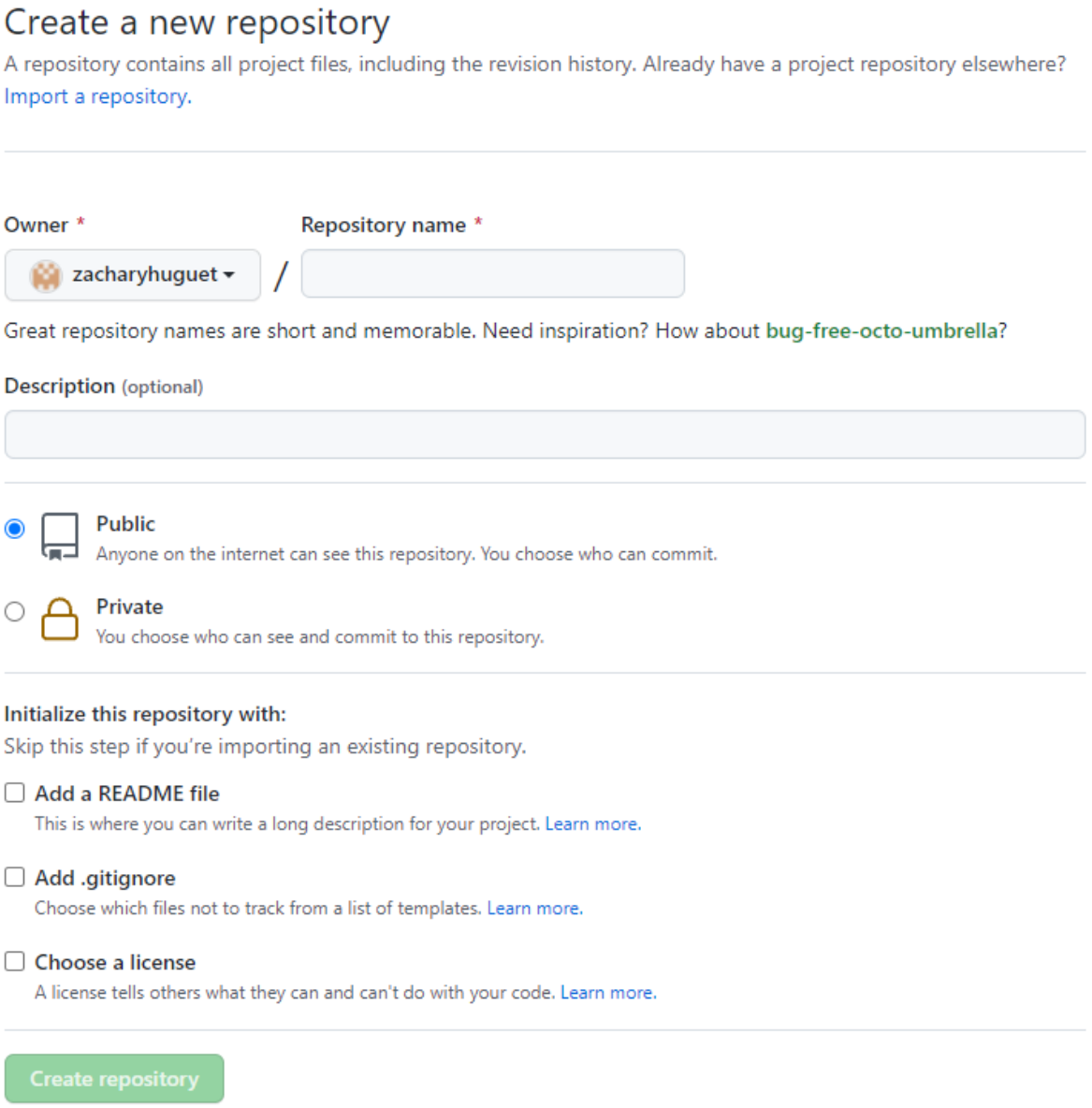
Utilisation Composer et Twig

Création d’un dépôts GitHub

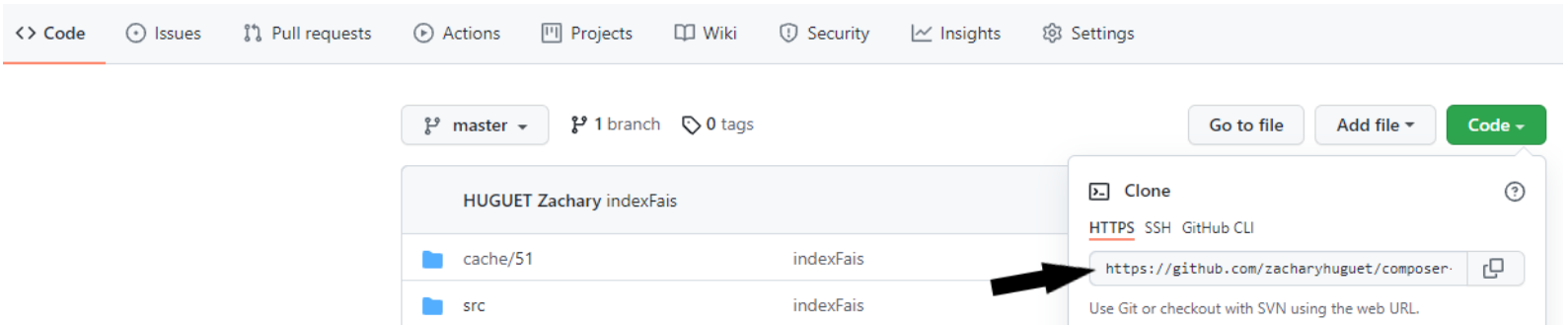
On va sur https://github.com/ ensuite on va sur sur son profil et on clique sur “Your repositories “pour créer un dépôt.

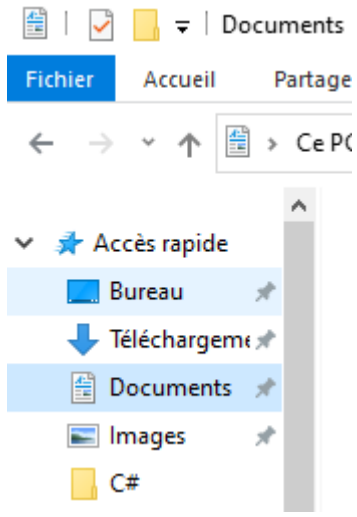




Pour la création on indique le nom du dépôt et on le met en publique pour qu'il soit visible par tous le monde.

ensuite on accède au dépôts en cliquant dessus puis on va cliquer sur "Code" (le bouton vert) pour copier le lien et cloner le dépôts

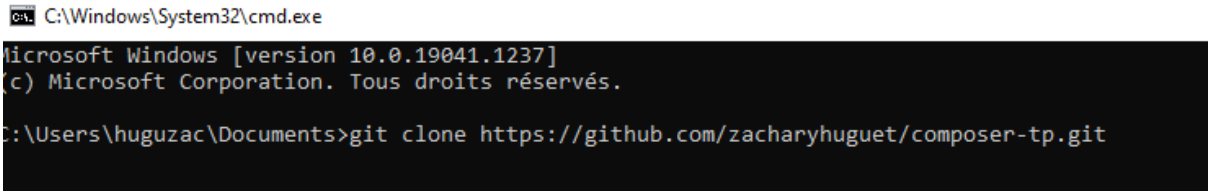




On va ensuite dans ces documents

Puis dans la barre pour accéder le chemin on tape “CMD”

Dans ce CMD on tape la commande “git clone (+ le lien que vous venez de cloner)“.



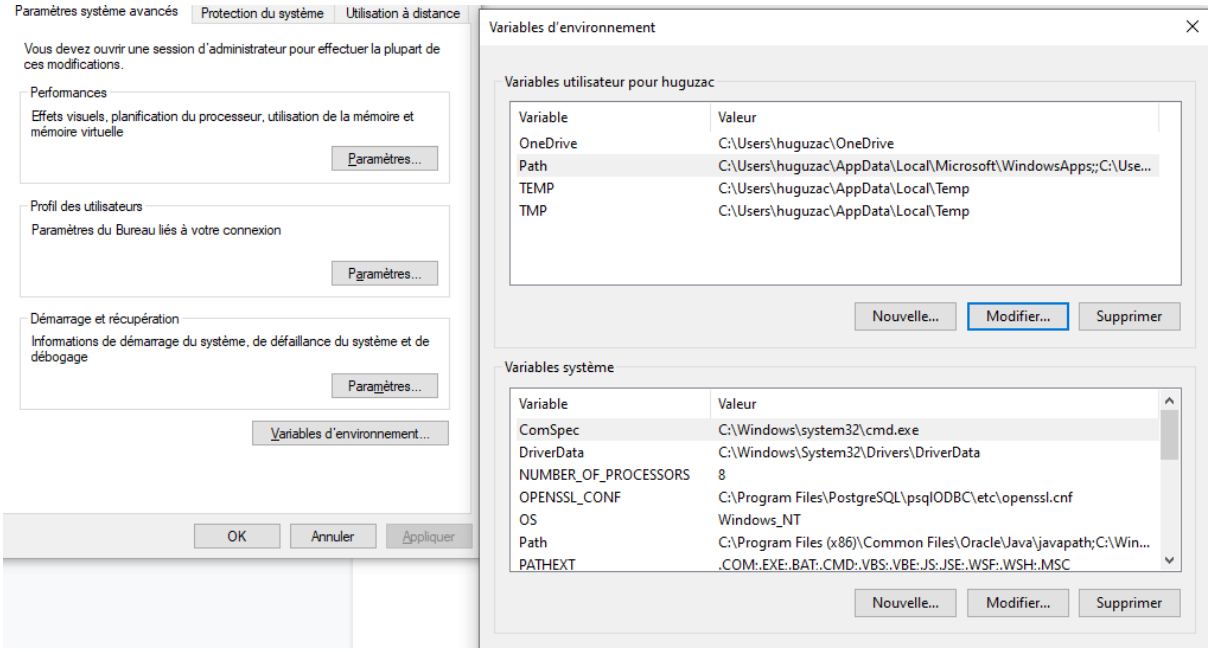
Le pc vient de cloner le dépôt github vide que vous avez créé précédemment.
Comme ça le dossier du pc et le dépôts github seront lié et vous pourrez mettre à jour vos document facilement avec les commandes :
-git add .
-git commit -m “(nom de la mise à jour)”.
-git push

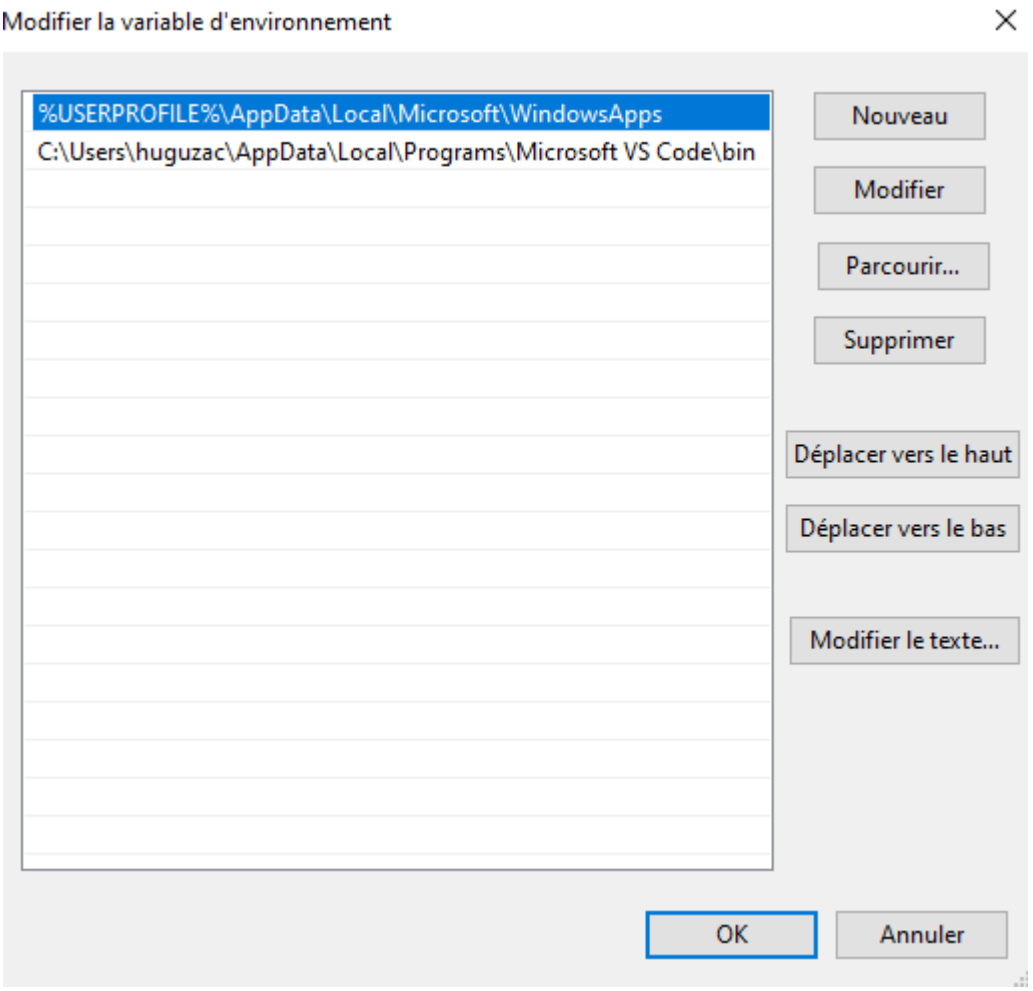
On doit aussi créer un fichier .gitignore pour indiquer à git lors d’un push quel fichier il ne doit pas prendre (fichier avec les mots de passe de connexion à la BD on crée donc un conf.sample pour indiquer comment on fait la connexion à la BD).

Installation php

Dernière version stable 8.0.11.
Pour télécharger php on va sur https://www.php.net/downloads

on indique le chemin du php.ini dans les variable d'environnement puis dans le path





Installation Symphony

https://symfony.com/download
On télécharge et exécute Symphony-Setup.exe

php C:\composer\v2\composer.phar create-project symfony/website-skeleton:"^4.4" essai1

On indique le chemin jusqu'à la localisation du composer.phar
On indique la version que l'on veut et le nom ensuite.

Installation Composer Monolog et Twig

On doit d'abord télécharger Composer https://getcomposer.org/download/
Moi j'ai installé la version 1.10.10

Ensuite on lance un document avec un logiciel de programmation (Visual Studio Code) et on on un nouveau terminal dans celui-ci.

Dans celui ci on commence par taper la commande :

- composer init

Ensuite on indique les champ suivant :

- Package name : VOTRE-NOM/VOTRE-NOM-DE-DÉPÔT
- Description : Small project to learn the composer tool
- Author : VOTRE-NOM <VOTRE-ADRESSE-EMAIL>
- Minimum Stability : stable
- Package Type : project
- License : MIT
- Would you like to define your dependencies : no
- Would you like to define your dev dependencies : no
- Do you confirm generation : yes

Après ses champs renseigner on tape la commande :

- composer install

Ainsi Composer va nous générer des documents comme composer.json

Ensuite on installe Monolog avec la commande :

- composer require monolog/monolog

Monolog sert à tracer le bon fonctionnement d'un code (savoir si tout marche bien et si non renvoi un erreur) il y à plusieurs :

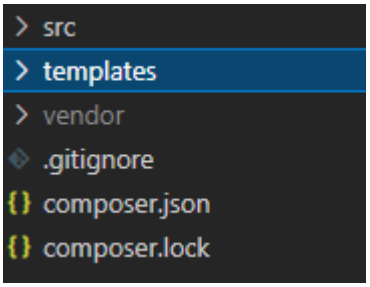
- DEBUG
- INFO
- NOTICE
- WARNING
- ERROR
- CRITICAL
- ALERT
- EMERGENCY

Et on installe aussi Twig avec la commande:

- composer require "twig/twig"

A la fin le composer.json doit ressembler a quelque chose comme ça.

Après, on créer à la racine un dossier scr pour contenir les fichiers sources de notre projet PHP.
et on crée aussi un dossier templates pour contenir les fichiers de nos modèles de pages HTML.



Ensuite on doit ajouter la ligne autoload dans le composer.json:

"autoload": {
"psr-4": { "App\\": "src/" }
}

Pour indiquer ou il iras chercher les fichiers PHP

Après, dans le dossier templates on crée le un fichier html.twig

dans l'index.php:

```php
echo $twig->render(
    'user/index.html.twig',
    [
        'users' => $users,
        'title' => "Titre de ma page très géniale",
    ]
```

et dans le html.twig:

```
<body>
    <h1>Logiciel - {{ title }}</h1>

    <ul>
    {% for user in users %}
        <li>{{ user.email }}</li>
    {% endfor %}
    </ul>
```

On voit que l'on initialise des variables dans le PHP et que l'on peut les appeler dans html.twig sans utiliser les balise pour ouvrir et fermer le php tout le temps.

Pour lancer le localhost on fait la commande:

- php -S localhost:8000 -t src

on ajoute le "-t src" pour dire que les fichier de base à afficher sont dans le dossier src.

Twig

Les Fonctionnalité de twig:

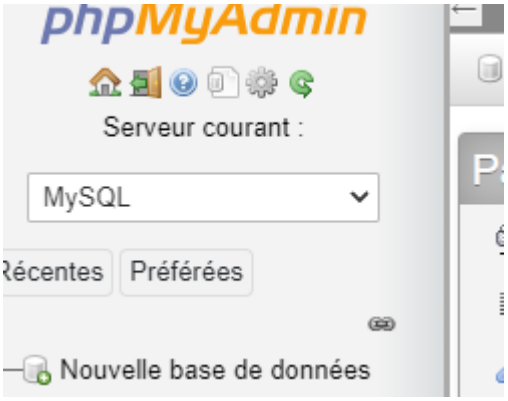
- {{ variable }} : affiche quelque chose
- {% commande %} : fait quelque chose
- {# commentaire #} : n'affiche rien et ne fait rien

Exemples de commandes :

- Pour l'affichage: <h1>Logiciel - {{ title }}</h1>
- {% for user in users %} -> boucle
- {# Un commentaire #}

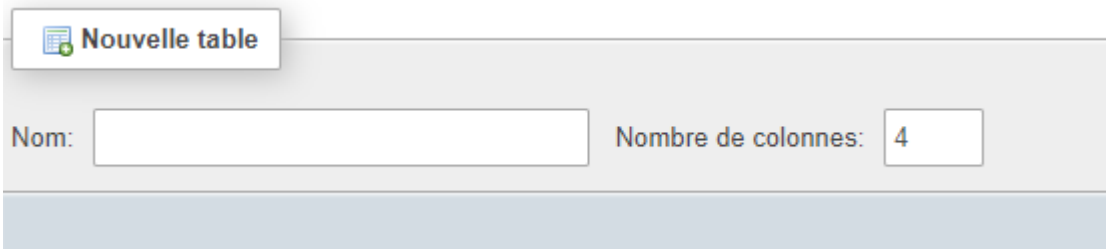
Création d'une base de donnée

Moi j'ai créé ma base de donnée avec phpmyadmin de wamp



On clique donc sur "Nouvelle base de données".

Ensuite on lui indique son nom et son type de langage (nous utf8)



Après on lui indique le nom de la table et le nombre de colonne que l'on veut dedans
La pour la gestion d'utilisateur je vais en mettre 4:

- id (clée primaire)
- email (varchar)
- password (varchar) -> sha1 pour crypter le mot de passe
- groupe (varchar)

On peut maintenant créer un utilisateur.
et renseigner au php comment se connecter à la base de données.

pour ça on créer un conf.php dans lequel on rentre:

confsample.php

```php
<?php
$dsn = 'mysql:dbname=NomDeLaBase;host=Ip';
$user = 'UnUtilisateur';
$password = 'UnMotDePasseTrèsSecurisé';
```

PHP Classe et gestion d’utilisateur

wamp64 > www > php-poo > archer.php

```php
<?php

final class Archer extends Personnage
{

    public function attaquer(Personnage $persoAFrapper): Personnage
    {
        $persoAFrapper->_degats += $this->_force;
        return $this;
    }

    public function insulter()
    {
        print("</br> tu vise comme un pied! ");
    }

}
```

Ici on creer une classe Archer avec un final qui veut dire qu’elle n’aura pas d’héritage on lui donne 2 méthodes attaquer et insulter

On crée ensuite un fichier PersonnageManager pour gérer les personnages

```php
class PersonnagesManager
{
    private $_db;

    public function __construct(PDO $db)
    {
        $this->setDb($db);
    }

    public function setDb(PDO $db): PersonnagesManager
    {
        $this->_db = $db;
        return $this;
    }
```

Dedans on met le Constructeur qui aidera à la création de nouveaux personnages.

On crée après un nouveau fichier avec une classe Personnage ou l'on renseigne tous les attributs d'un personnage et les assesseurs et les guetteurs pour chacun des attributs.

```php
abstract class Personnage
{

    // Déclaration des attributs et méthodes ici.
    private $_id = 0;
    private $_nom = 'Inconnu'; // Son nom, par défaut 'Inconnu'.
    protected $_force = 50; // La force du personnage, par défaut à 50.
    private $_experience = 1; // Son expérience, par défaut à 1.
    private $_degats = 0; // Ses dégats par défaut
    private $_niveau = 0;
    private $_classe = 0;
    private $_poche = 50;
```

```php
public function setNom(string $nom): Personnage
{
    if (!is_string($nom)) // S'il ne s'agit pas d'un texte.
    {
        trigger_error('Le nom d\'un personnage doit être un texte', E_USER_ERROR);
        return $this;
    }
    $this->_nom = $nom;
    return $this;
}

public function getNom(): string
{
    return $this->_nom;
}
```

Utilisation d’un bootstrap

On cherche un thème souhaité sur https://getbootstrap.com/
Une fois trouvé on copie et on le met dans de le <head>