Hugo JENOUVRIER BTS SIO 2

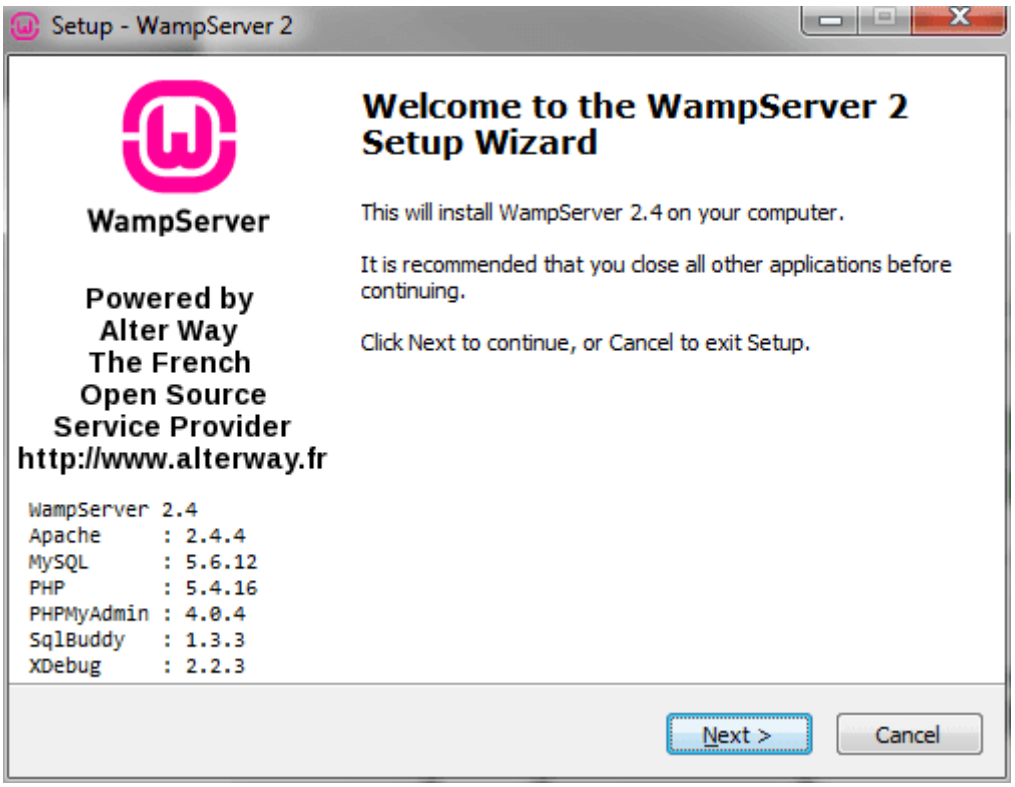
Gestion des utilisateurs avec Composer et Twig

I / Les installations requises

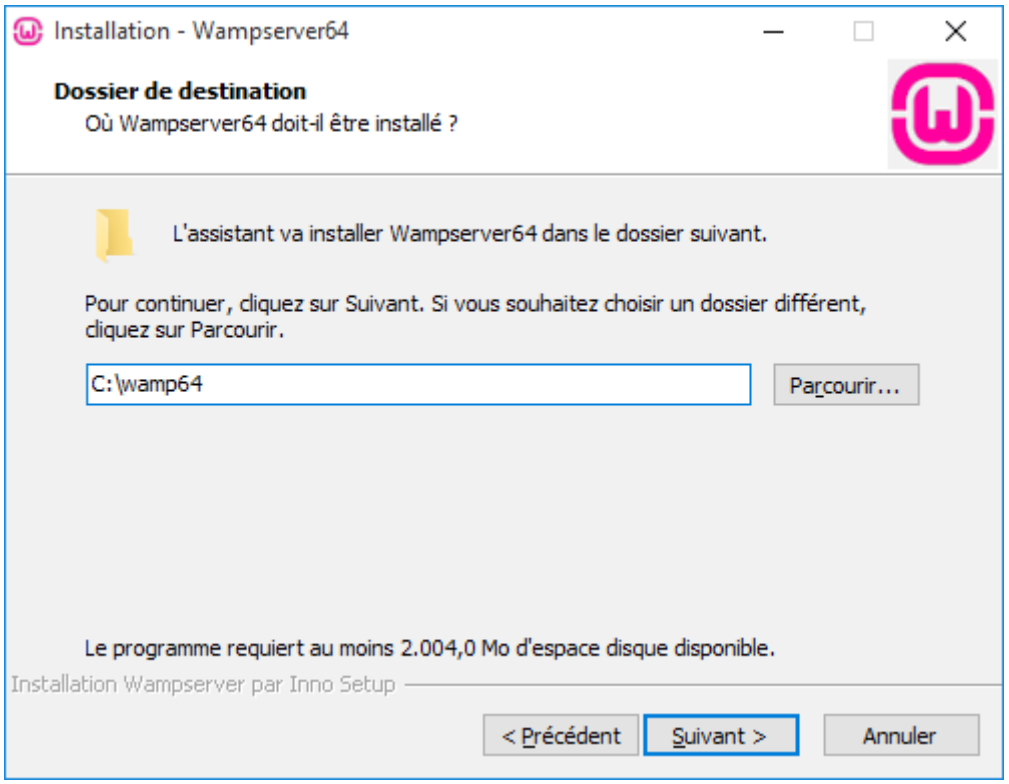
a) Comment installer PHP et MySQL

Pour installer PHP et MySQL vous pouvez directement installer wamp sur votre machine. https://www.wampserver.com/fr/. Suite à ça vous aurez alors tous les packet utiles pour PHP et MySQL.

Nous allons commencer par installer Wamp64.



Continuer les étapes de l'installation. Une fois arrivé sur cette page

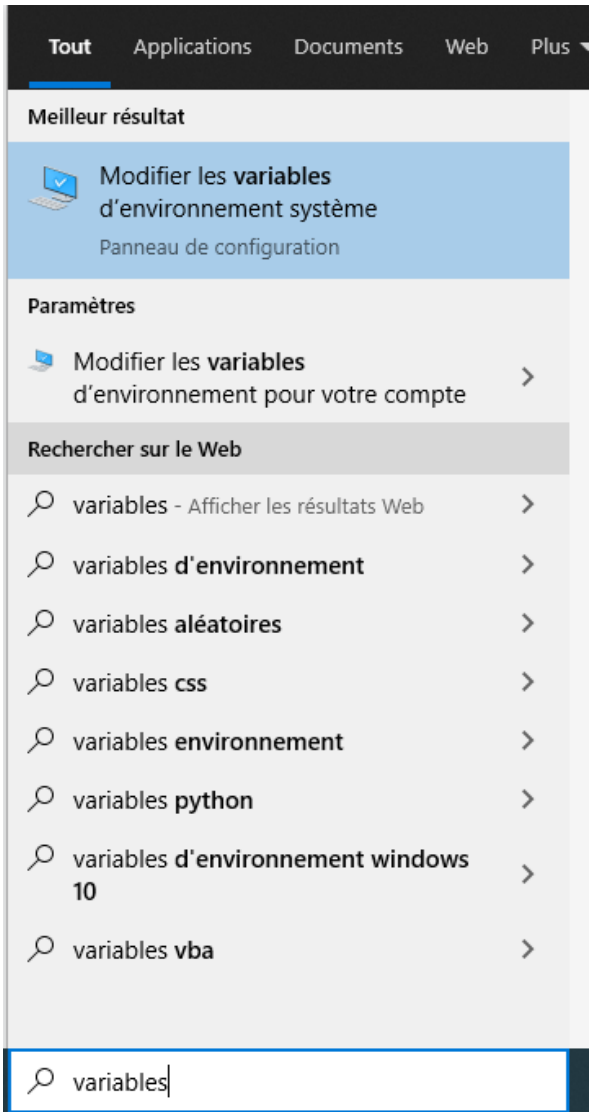


Laissez le dossier d'installation dans le Disque C: pour éviter tous les problèmes qui peuvent survenir.
Et continuer l'installation, une fois l'installation faite vous avez alors php et MySQL d'installer sur votre machine.

Vous aurez alors accès à MySQL mais pas à PHP il faut l'ajouter alors dans une variable d'environnement :

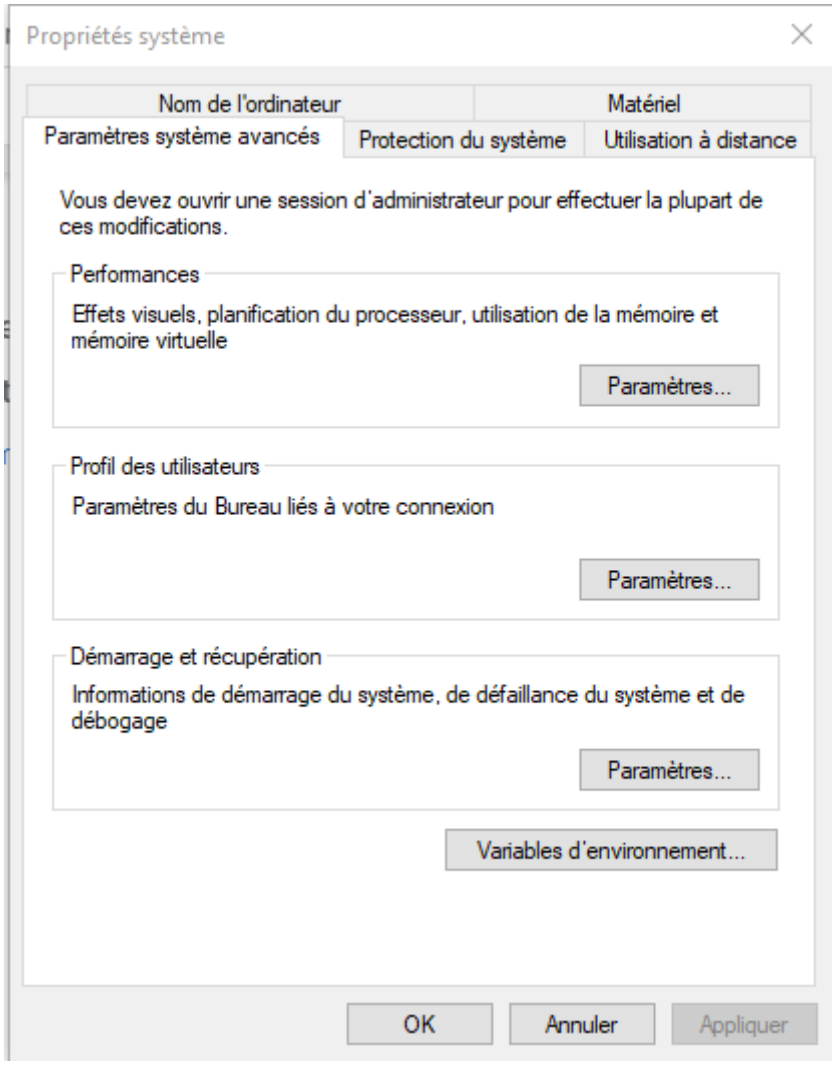
La mise en place d'une variable d'environnement est disponible sur la suite des pages.

Recherchez les variable d’environnement sur votre machine ci-dessous :

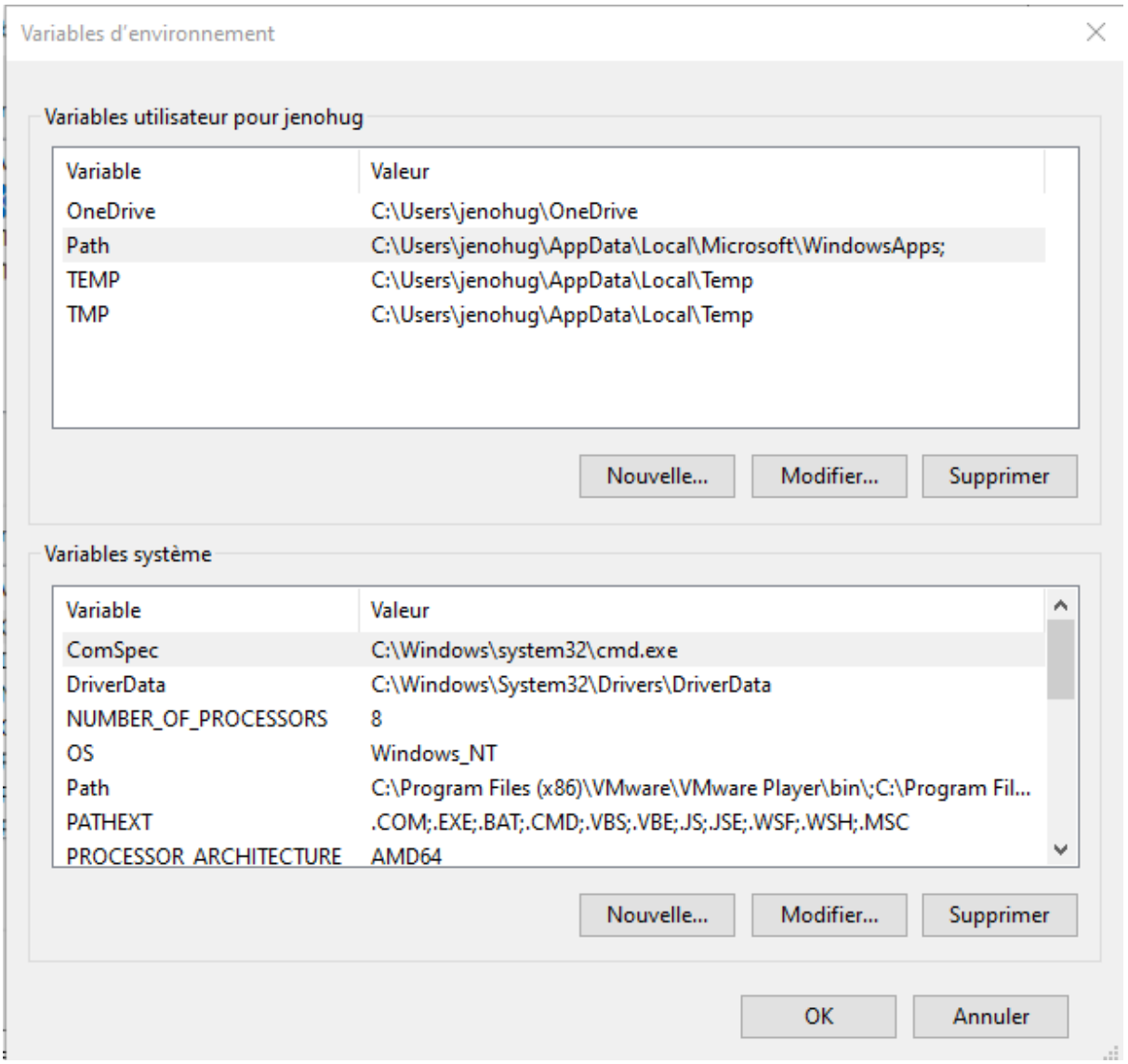


Ensuite ouvrez : Modifier les variables d’environnement système.

Une page s’ouvrira comme celle-ci :

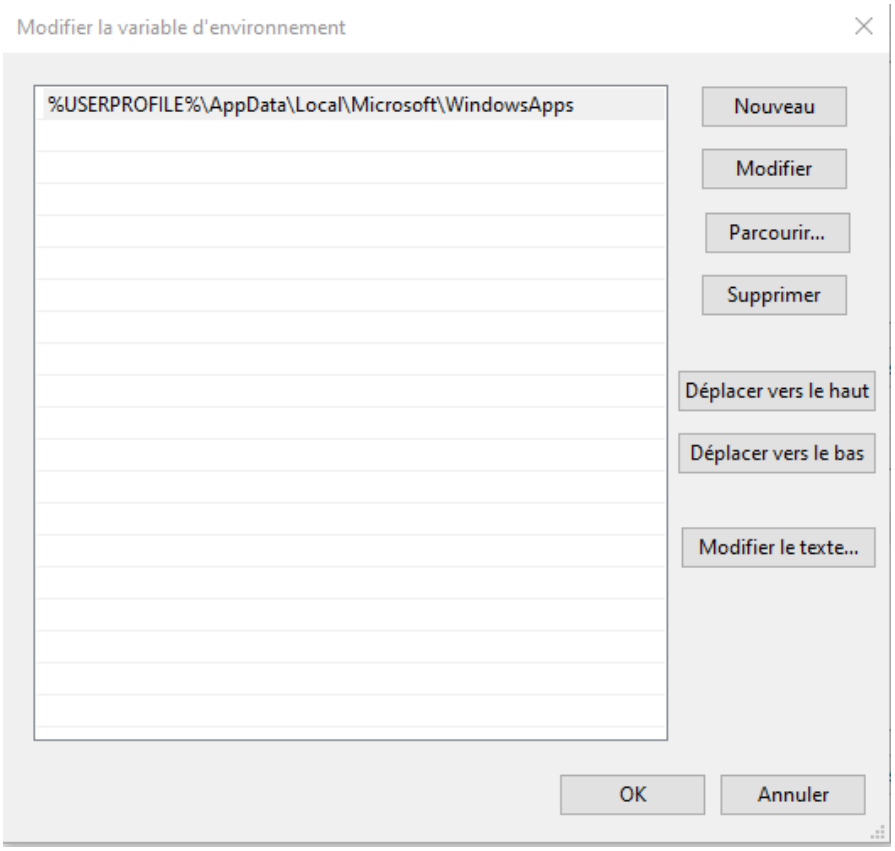


A partir de la, cliquez sur le bouton Variables d'environnement et ensuite une nouvelle page va s'ouvrir :



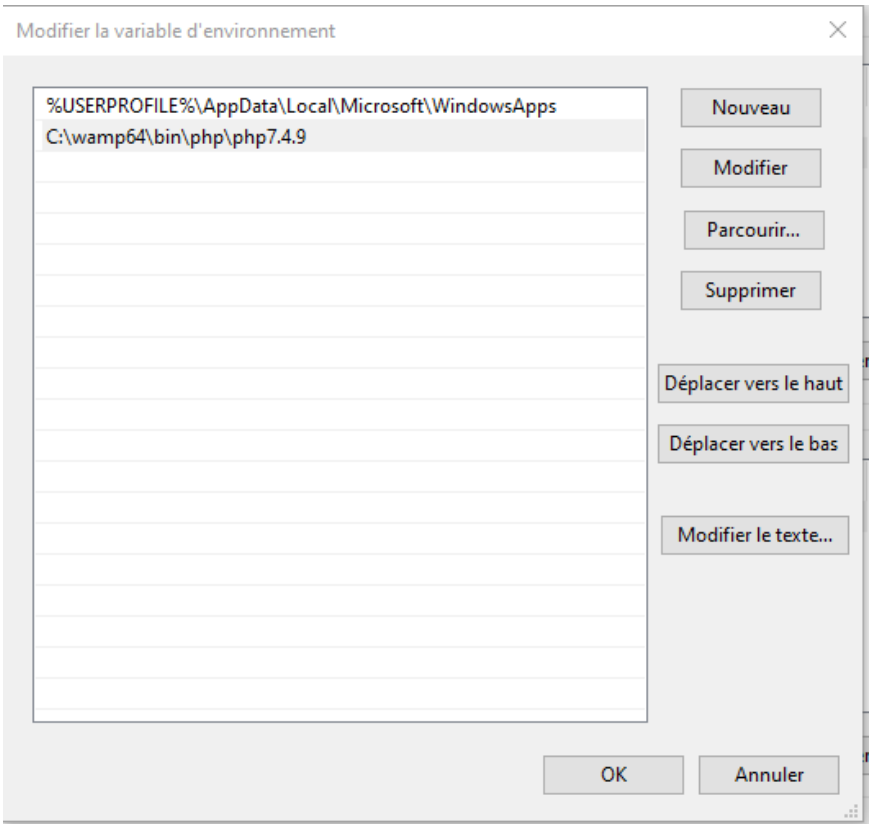
Sur cette page vous ne devez avoir que 4 variables utilisateur tel que **OneDrive**, **PATH**, **TEMP** et **TMP**

Faite un double clique sur **PATH** et vous allez arrivez sur cette page :



Cliquez alors sur **Nouveau** et mettez le chemin d'accès à php qui se trouve alors dans votre dossier C:\wamp64\bin\php\"votre_version_php".

Ici pour moi c’est php7.4.9 :



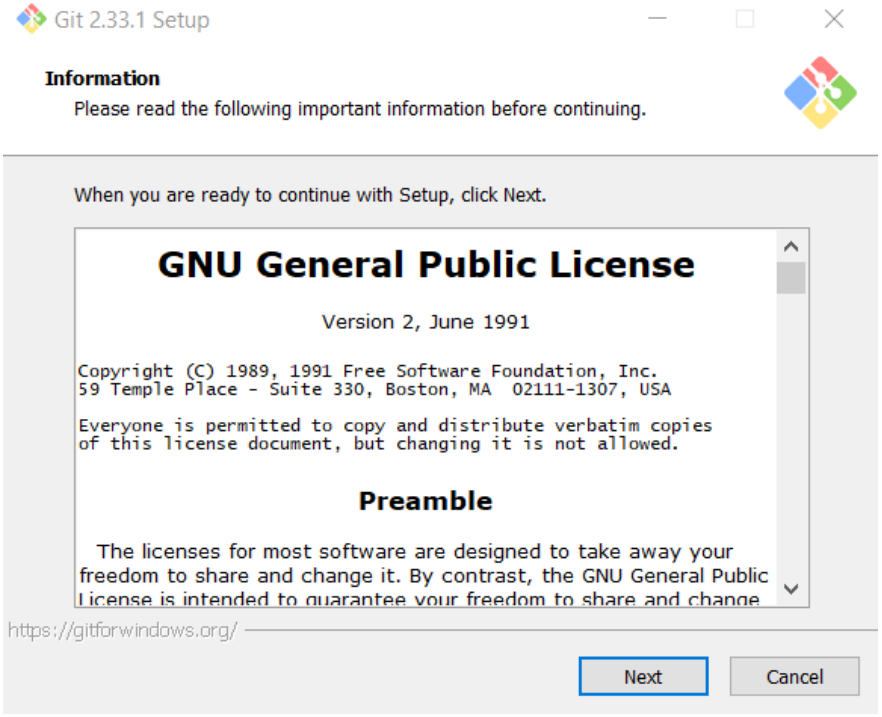
Et à partir de la faites Ok sur toutes les pages. Ensuite vous avez ajouté votre variable d’environnement.

Pas besoin de faire la même manipulation pour MySQL car il est directement installé avec Wamp64.

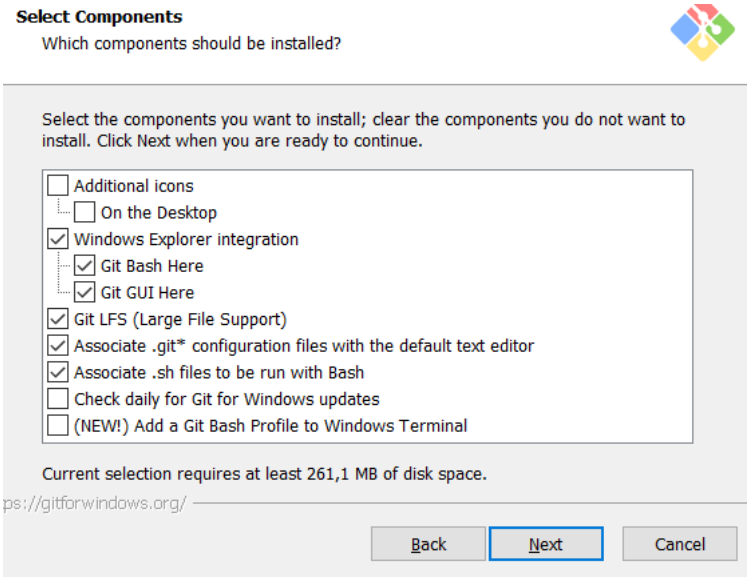
b) Installer GIT

Pour installer GIT 2.33.1 allez sur le lien suivant : https://git-scm.com/downloads,
Vous allez alors arriver sur une page web et cliquer sur Windows si votre machine est un Windows.

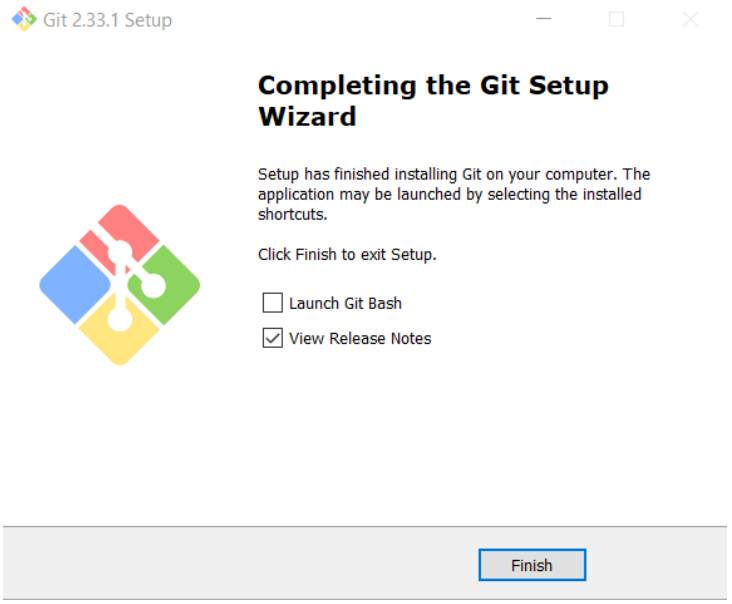
Le téléchargement de l'exécutable va commencer, une fois terminé ouvrez le.



L’installateur va alors s’ouvrir. Cliquez alors sur next vous allez devoir choisir votre chemin d’installation. De préférence laisser celui mit automatiquement.



Vous allez arriver sur cette page. Vous pouvez si vous le souhaitez mettre un icon sur votre bureau mais ce n’est pas forcément nécessaire. Cliquez alors sur Suivant jusqu'à ce que l’installation démarre.



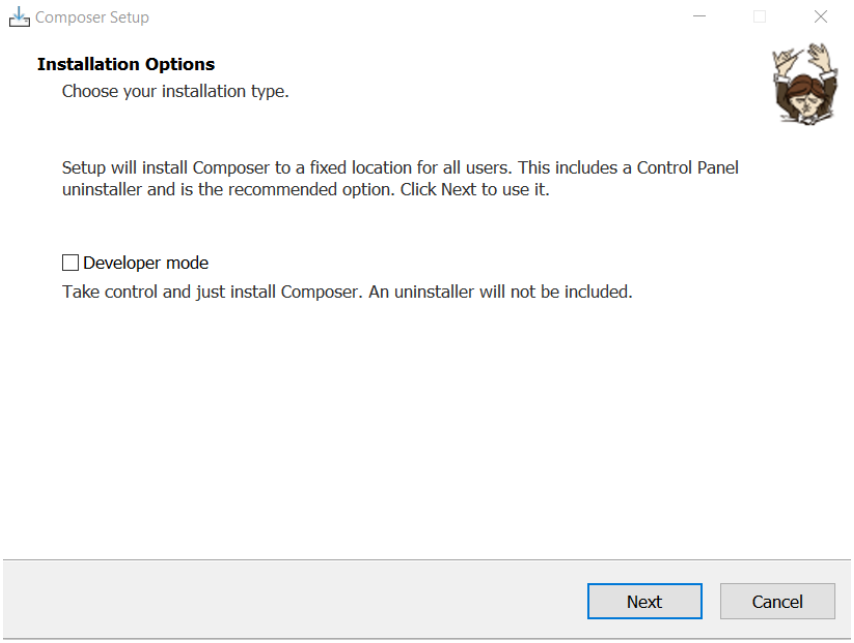
Une fois l’installation finie, ceci apparaîtra. Cliquez sur finish et GIT 2.33.1 sera alors installé.

c) Installer Composer

Pour installer Composer rendez vous sur : https://getcomposer.org/download/, et cliquez sur Composer-Setup.exe

Download and run Composer-Setup.exe - it will install the latest composer version whenever it is executed.

Une fois cela installé, un installateur pourra être exécuté.
Ouvrez le et faites l'installation.

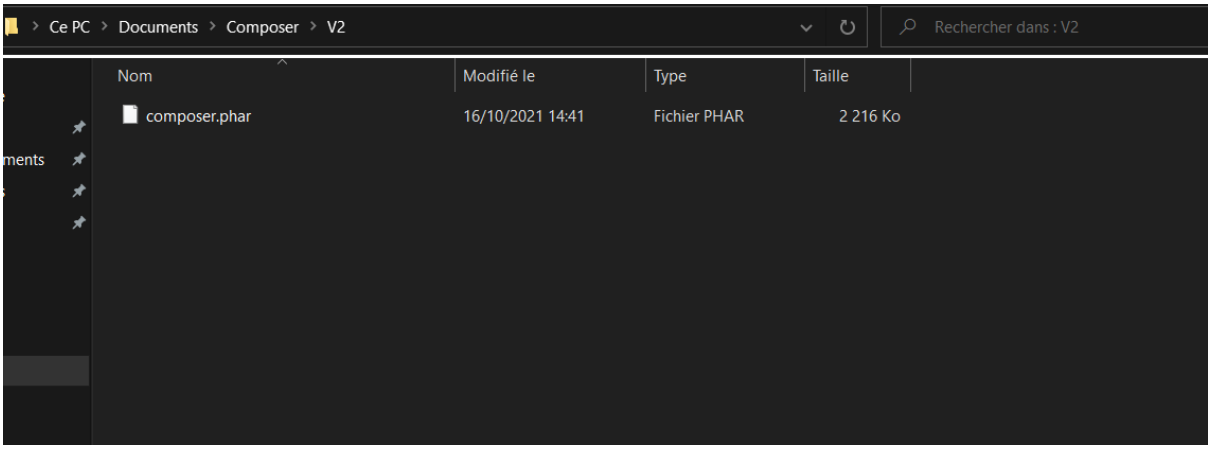


Ensuite vous devrez choisir votre version de PHP, prenez alors la 7.4.9

Une fois cela fait vous pouvez alors mettre votre proxy si vous en avez un, sinon faite next puis Install.

Ensuite, retournez sur le site de Composer. Et prenez la version stable du Composer.phar ainsi que par exemple la version 2.1.9 de Composer.phar

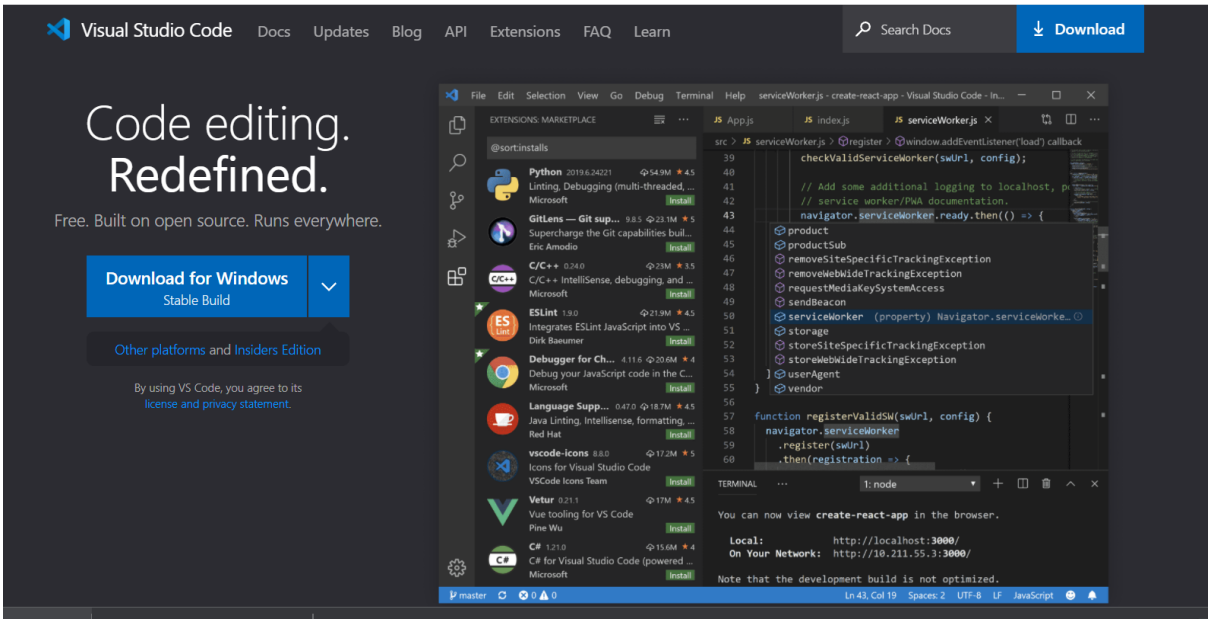
Une fois téléchargez créez un document ou vous le souhaitez que vous appellerez Composer. Et dans ce Document vous allez créer deux répertoires avec le nom V1 et V2. Vous mettrez ainsi la version stable dans le dossier V1 et l'autre version dans le dossier V2. Comme ci-dessous :



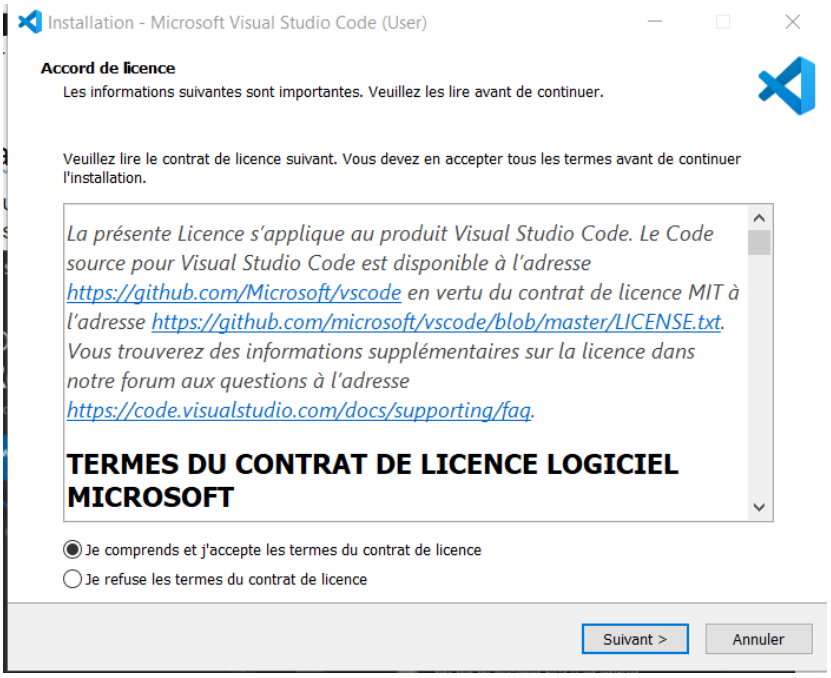
Composer est donc installé sur votre machine.

d) Installer Visual Studio Code

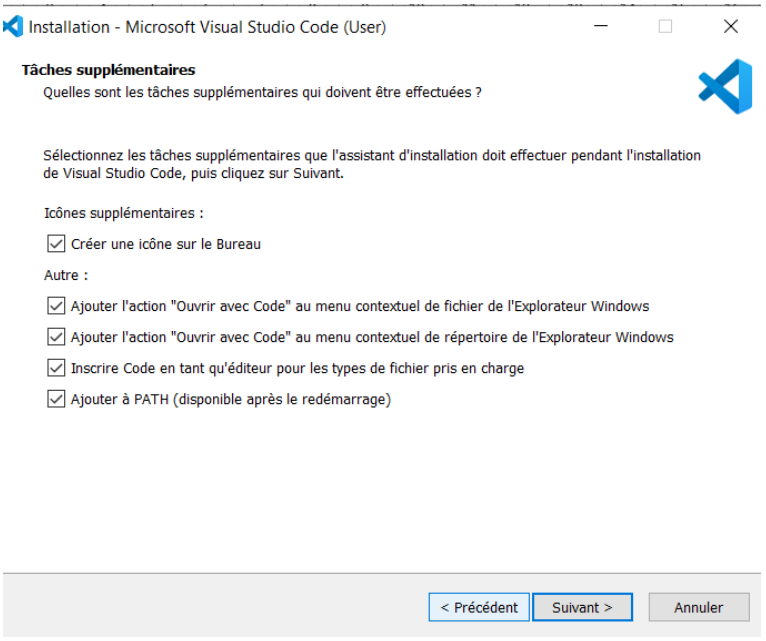
Pour installer Visual Studio Code rendez vous sur :
https://code.visualstudio.com/
et cliquez sur Download for Window. Pour avoir alors la version stable.



Le téléchargement du setup va alors démarrer.



Accepter les termes du contrat et poursuivez l'installation



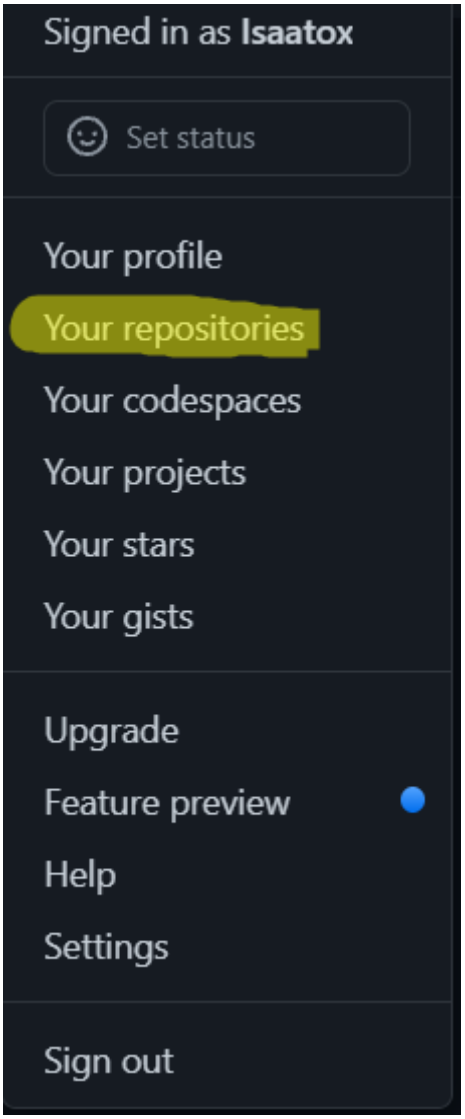
Choisissez alors ce que vous voulez avec Visual Studio Code, tel que l'icône sur le bureau, pouvoir ouvrir vos dossiers avec un clique droit et autre. Cliquez sur suivant et installez.

Une fois l'installation terminée, Visual Studio Code est donc installé sur votre machine.

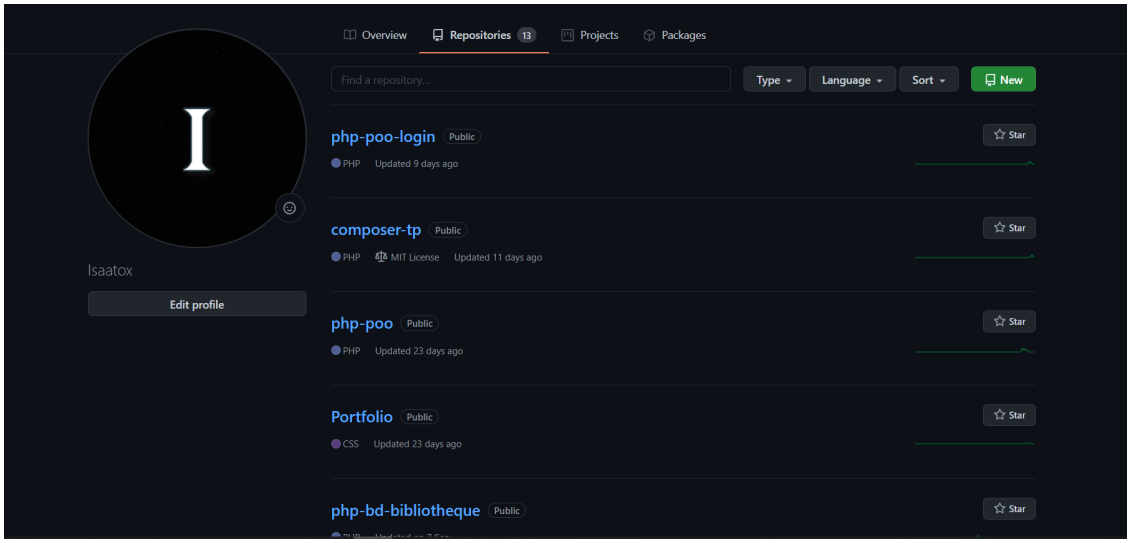
e) Installer HeidiSQL

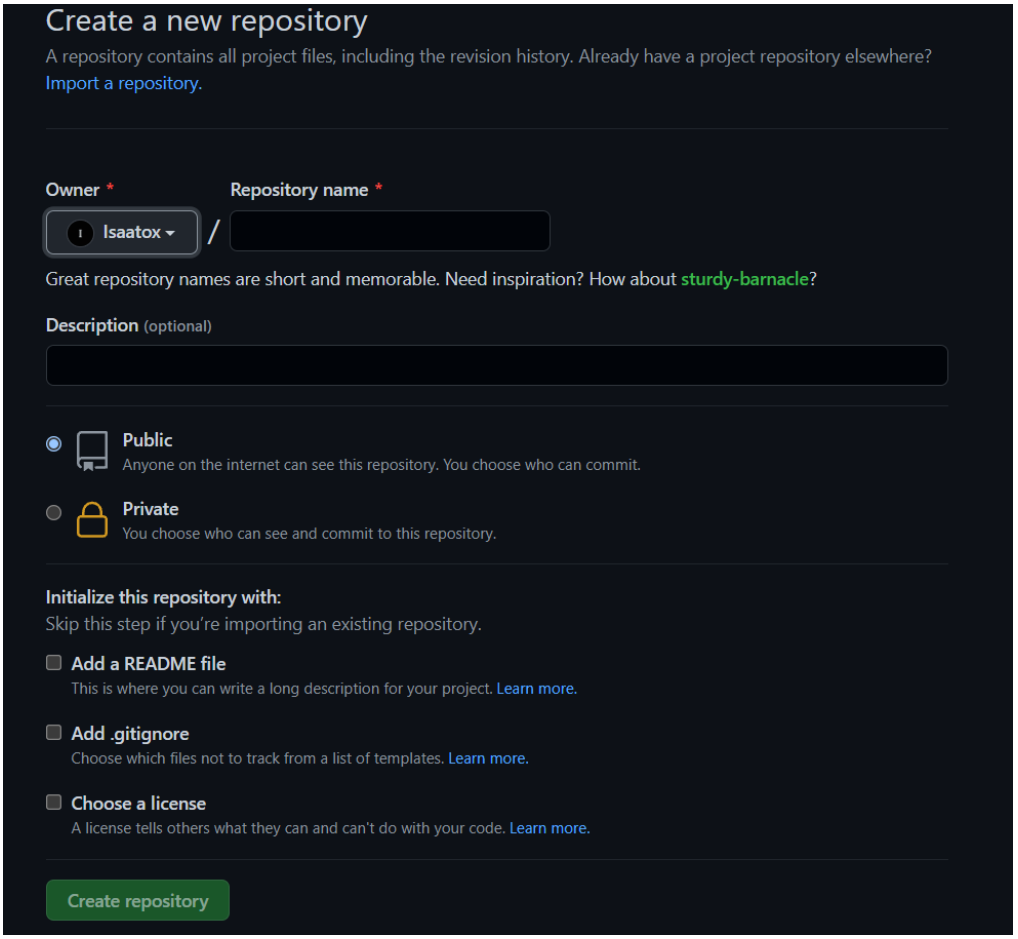
II/ Créer un dépot publique Git

Allez sur GitHub : https://github.com/ et connectez vous à votre compte Git.
Cliquez sur votre logo et ensuite ouvrez : Your Repositories :



Vous arriverez alors sur cette page :
Cliquez alors sur le bouton New pour créer un dépôt public.
Et une nouvelle page s'affiche comme ci-dessous.

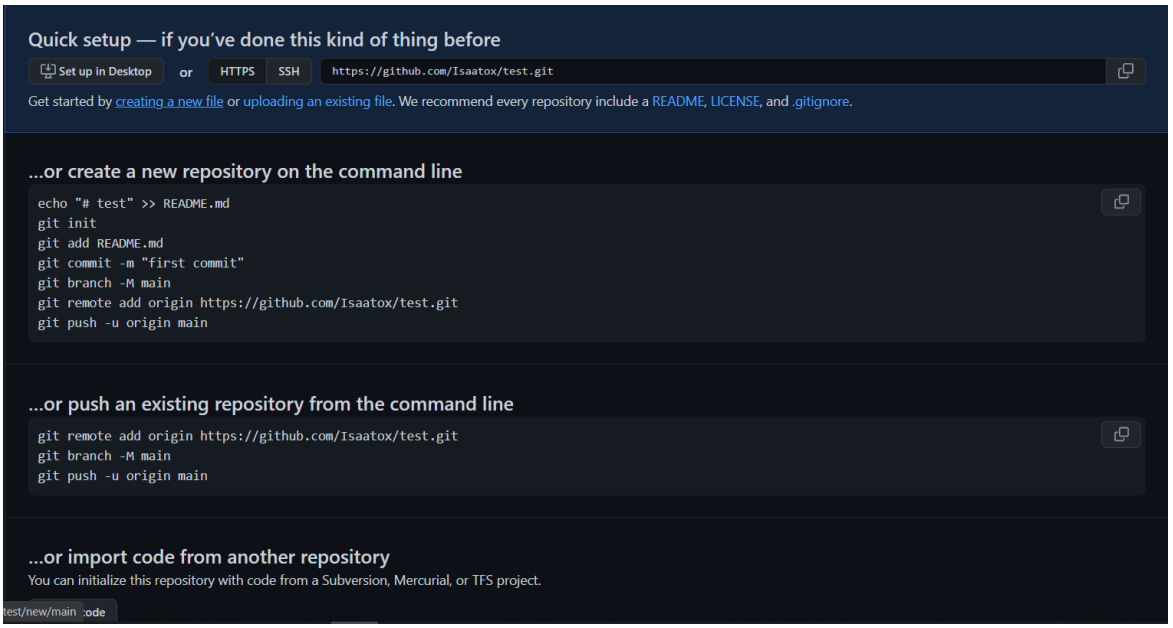




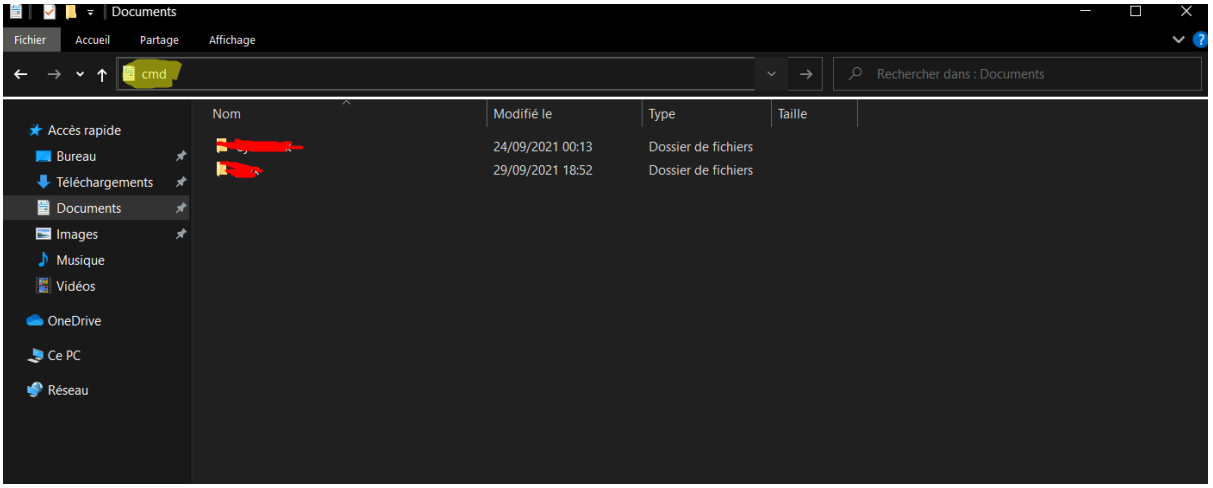
Mettez un nom et une description si vous le souhaitez.

Ensuite laissez cochez public. Vous pouvez aussi ajouter un ReadME, un .gitignore et une licence si vous le souhaitez.

Quand cela est fait vous pouvez alors cliquer sur le bouton Create repository. Alors si votre dépôt est bien créer cette page s'affichera :



Vous pouvez alors copier le lien et allez dans vos documents pour avoir le dépôt sur votre machine. Pour cela ouvrez vos document et écrivez cmd :



Ainsi un Invite de commande va s'ouvrir.
Et écrivez la commande git clone "le_lien_du_dépôt"

Votre dépôt sera alors accessible depuis votre machine.

III/ Création d'un projet PHP avec Composer

Pour créer un projet PHP avec Composer créer d'abord un dépôt github public que vous appellerez par exemple Composer-TP.

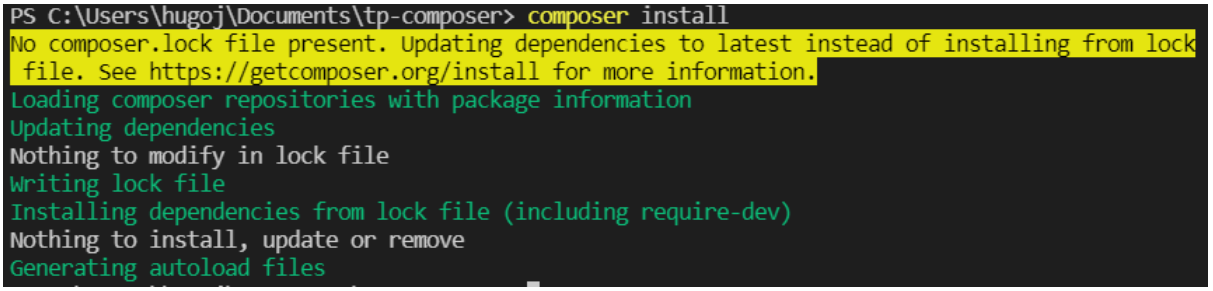
Vous sélectionnerez alors le ReadMe, une licence de Type MIT, et un .gitignore avec composer dedans.

Clonez votre répertoire dans vos Documents et ouvrez le avec Visual Studio Code.

Une fois votre répertoire ouvert avec Visual Studio Code, ouvrez un terminal et écrivez **composer init :**

Package name : VOTRE-NOM/tp-composer
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

Une fois cela effectué faite la commande **composer install**



Quand cela est fait, vérifiez votre fichier composer.json.

Le fichier composer.json peut être considéré comme une liste de recherche pour composer .

Grâce à cette liste, composer ne téléchargera que les packages (dépendances) de votre projet qui sont mentionnés dans ce fichier. Il vérifie également la compatibilité des versions des packages avec votre projet.

Votre projet Composer est alors mis en place sur votre machine.

IV / Mis en place de la traçabilité avec Monolog

Pour mettre en place Monolog avec Composer ouvrez le terminal et faites : **composer require monolog/monolog**

L'installation des dépendances peut être plus ou moins longue.

Quand l'installation est finie allez sur composer.json et vous verrez que de nouvelles lignes sont apparues. Dans la ligne "monolog/monolog": "^2.3" , l'accent circonflexe (^) est destiné à permettre toutes les mises à jour sans changement majeur.

Créer ensuite le dossier *src*, ce dossier va contenir les fichiers sources de notre projet.
Créer aussi le dossier *log*, qui va contenir le fichier qui tracent les messages générés lors de l'exécution de notre projet (**app.log**).

Dans le fichier composer.json ajouter les lignes suivantes (ajouter si besoin une virgule) :

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

La traçabilité **Monolog** est alors mise en place dans votre projet.

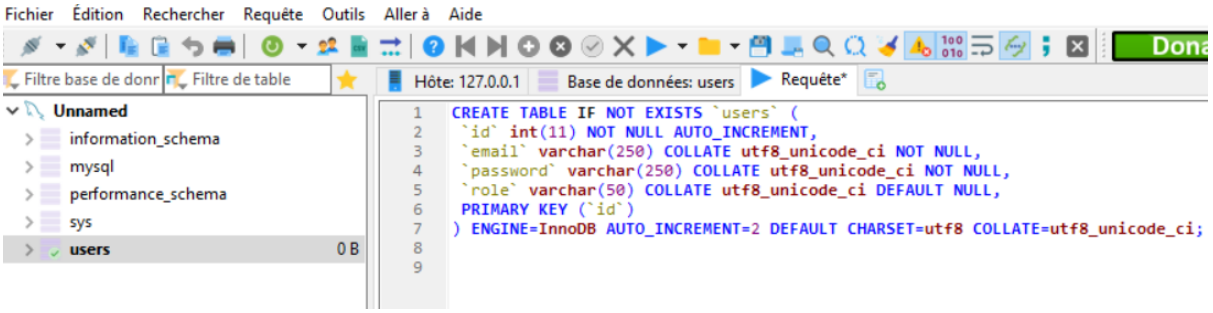
V/ Créer la base de données

Pour commencer ouvrez votre logiciel pour gérer les bases de données. Pour ma part ce sera HeidiSQL mais vous pouvez aussi utiliser PHPmyadmin.

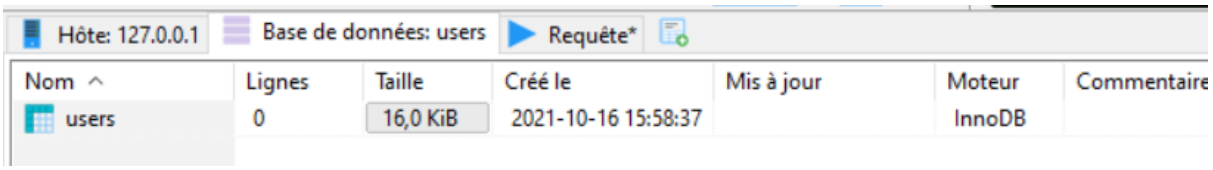
Nous allons créer une base se nommant Users. Dans cette base nous allons rentrer cette requête :

```
CREATE TABLE IF NOT EXISTS `users` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `email` varchar(250) COLLATE utf8_unicode_ci NOT NULL,
 `password` varchar(250) COLLATE utf8_unicode_ci NOT NULL,
 `role` varchar(50) COLLATE utf8_unicode_ci DEFAULT NULL,
 PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8
COLLATE=utf8_unicode_ci;
```

Cela nous permet d'avoir ce qu'il faut pour gérer les utilisateurs.



Mettez alors la requête dans l'onglet requête et exécutez la. Si cela a bien fonctionné vous devriez voir votre table dans la base de données (actualisez la base en appuyant sur **F5**).



Voilà votre base de données est maintenant créée pour le projet.
Dans Visual Studio Code créer un fichier dans le dossier *src* se nommant **conf.php** et vous allez écrire dedans :

```php
<?php

$dsn = 'mysql:dbname=users;host=127.0.0.1';
$user = 'root';
```

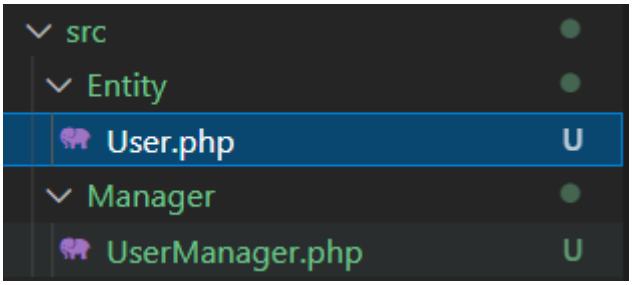
```
$password = 'aVerySecurePassword';
```

Ceci va permettre la connexion à la base de données.

VI/ Création Entity et Manager

Retourner dans Visual Studio Code, allez dans le dossier *src* et créer 2 dossier tel que : *Entity* et *Manager*. Dans le dossier *Entity* créer le fichier **User.php** et dans le dossier *Manager* créer le fichier **UserManager.php**

Comme ci dessous :



a) Entity

Dans le fichier *User.php* utiliser le **namespace App\Entity** et écrivez **Use App\Entity\User;**

Ensuite créer la classe User en déclarant les attributs id, email, password et rôle en privé. comme ci dessous :

```
namespace App\Entity;

use App\Entity\User;

class User {
    private $_id;
    private $_email;
    private $_password;
    private $_role;
```

Créer aussi 2 méthodes publiques : la méthode constructeur et la méthode hydrate.

```php
public function __construct(array $ligne)
{
    $this->hydrate($ligne);
}

public function hydrate(array $ligne)
{
    foreach ($ligne as $key => $value) {
        $method = 'set' . ucfirst($key);
        if (method_exists($this, $method)) {
            $this->$method($value);
        }
    }
}
```

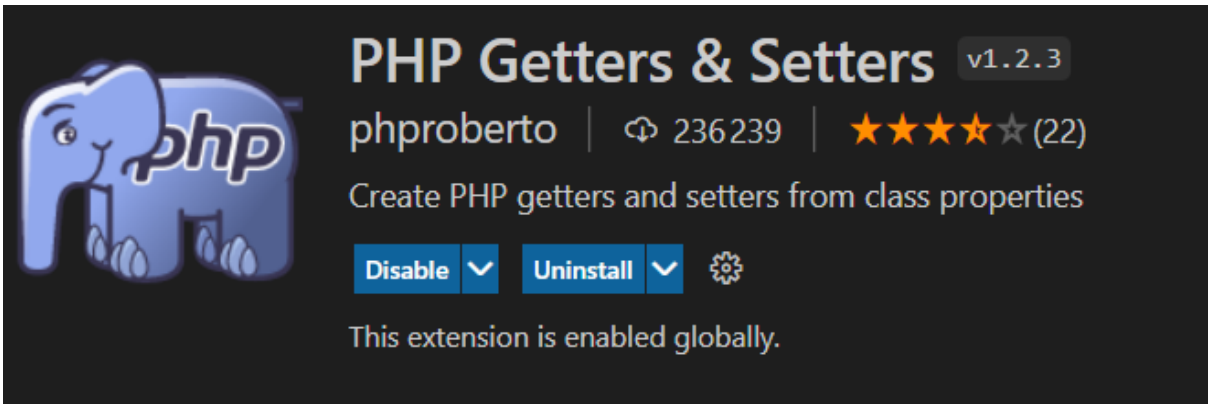
Déclarer par la suite le getters et setters de chaque attribut private.

```php
/**
 * Get the value of _email
 */
public function getEmail()
{
    return $this->_email;
}

/**
 * Set the value of _email
 *
 * @return  self
 */
public function setEmail($_email)
{
    $this->_email = $_email;

    return $this;
}

/**
 * Get the value of _password
 */
public function getPassword()
{
```

```
        return $this->_password;
    }

    /**
     * Set the value of _password
     *
     * @return  self
     */
    public function setPassword($password)
    {
        $this->_password = password_hash($password, PASSWORD_BCRYPT);

        return $this;
    }

    /**
     * Get the value of _role
     */
    public function getRole()
    {
        return $this->_role;
    }

    /**
     * Set the value of _role
     *
     * @return  self
     */
    public function setRole($_role)
    {
        $this->_role = $_role;

        return $this;
    }

    /**
     * Get the value of _id
     */
    public function getId()
    {
        return $this->_id;
    }
```

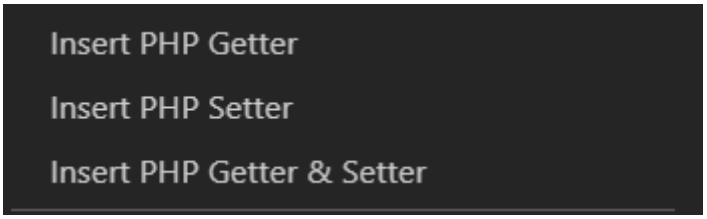
```php
/**
 * Set the value of _id
 *
 * @return  self
 */
public function setId($_id)
{
    $this->_id = $_id;

    return $this;
}
```

Vous pouvez soit les écrire à la main, soit utilisez une extension proposez par visual studio code qui s'appelle PHP Getters & Setters



Après avoir installé cette extension vous pouvez faire un clique droit sur votre attribut et alors cliquez sur Insert PHP Getters & Setters.

Mais vous pouvez aussi avoir juste le Getters ou le Setters.



b) Manager

Dans le fichier **UserManager.php** utiliser le **namespace App\Manager** et écrivez **Use App\Entity\User;**

Ensuite créer la classe UserManager en déclarant l'attribut db pour la base de données en privé. comme ci dessous :

```php
namespace App\Manager;

use PDO;
use Exception;
use App\Entity\User;

class UserManager
{
    private $_db;
```

Créer alors 2 méthode dont une constructeur et une autre setDB comme ci dessous :

```php
public function __construct(PDO $db)
    {
        $this->setDB($db);
    }

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

Le **Manager** vous permet de créer des méthodes telles que Update, Add, Delete, GetAll, GetOne … Pour gérer vos utilisateurs. Vous pouvez alors les créer en amont.

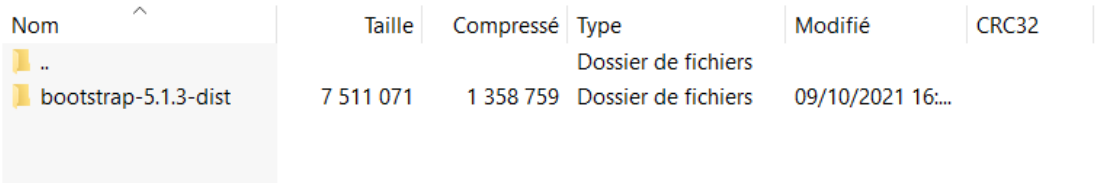
```php
public function add(User $user): bool
    {
        //Exécute une requête de type insert
    }

    public function delete(User $user): bool
    {
        //Exécute une requête de type delete
    }

    public function getAll(): array
    {
        //Exécute une requête de type select
    }

    public function getOne(int $id)
    {
        //Exécute une requête de type select
    }

    public function update(Personnage $perso): bool
    {
        //Exécute une requête de type UPDATE
    }
```

Vous avez alors votre Manager de prêt.

VII/ Mise en place d'une template Bootstrap

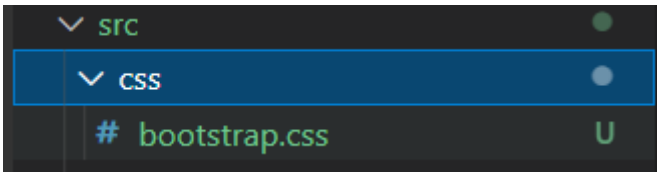
Pour mettre en place la template bootstrap rendez vous sur le site de bootstrap : https://getbootstrap.com/docs/5.1/getting-started/download/, et téléchargez le css

Ensuite un dossier .rar va être télécharger, ouvrez le.



Ouvrez ensuite le dossier bootstrap-5.1.3-dist et ouvrez le dossier css.
Copiez ensuite le fichier bootstrap.css

Dans Visual Studio Code, allez à la racine de dossier *src* et créer un nouveau dossier que vous appellerez *css*, dans ce dossier vous collerez alors le fichier **bootstrap.css**



Votre template CSS est alors mis en place.

VIII/ Mise en place de twig

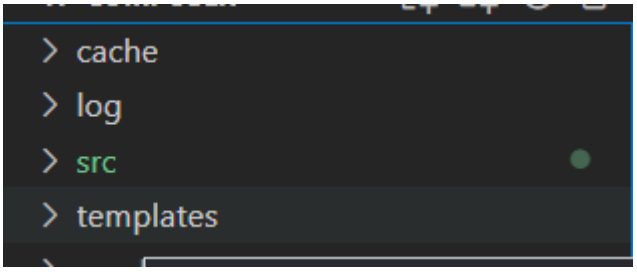
Pour mettre en place twig il faut installer le package de Twig en écrivant dans le terminal :

composer require "twig/twig"

L'installation peut prendre plus ou moins de temps.

Une fois cela fait, vous irez dans le composer.json et vous apercevrez qu'une nouvelle ligne est apparue.

Ensuite créer deux nouveau dossier : le dossier *templates* et le dossier *cache*



A savoir que Twig permet de séparer le code HTML du code PHP. De plus Twig utilise 3 type de syntaxe qui sont :

{{ ... }} : affiche quelque chose (une variable)
{% ... %} : fait quelque chose (une commande)
{# ... #} : n'affiche rien et ne fait rien (commentaire)

Par exemple :

pour une assignation :
{% set foo = 'bar' %}

pour une condition :
{% if i is defined and i == 1 %} ... {% endif %}

pour créer un compteur dans une boucle :
{% for i in 0..10 %} ... {% endfor %}

pour créer un tableau itératif :
{% set myArray = [1, 2] %}

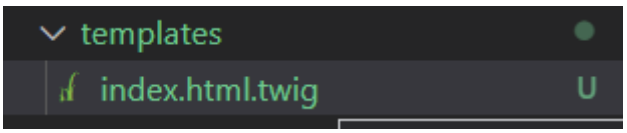
pour créer un tableau associatif :
{% set myArray = {'key': 'value'} %}

IX/ Création des templates

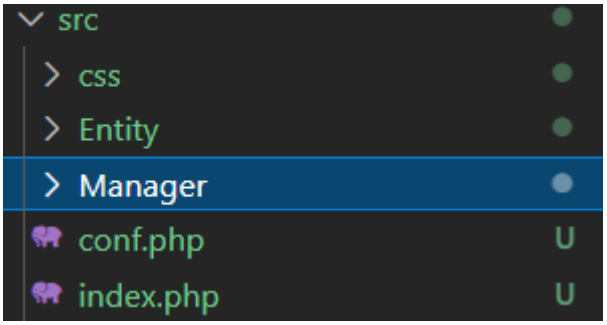
a) Index

Nous allons commencer par créer **l'index.php** ce qui va nous permettre de voir la liste des utilisateurs de notre projet.

Créer le template **index.html.twig** :



Ensuite créer le **index.php** dans le dossier *src* :



Dans le fichier **index.php** il faut appeler **autoload.php** avec require_once

```
require_once '../vendor/autoload.php';
```

Ensuite il faut préciser où se trouve les templates

```
$loader = new \Twig\Loader\FilesystemLoader('../templates');
```

Il faut aussi préciser qu'elle fichier nous allons utiliser :

```
use Monolog\Logger;
use Twig\Environment;
use App\Manager\UserManager;
use Twig\Loader\FilesystemLoader;
use Monolog\Handler\StreamHandler;
```

Par la suite il faut dire ou doit se trouver les log :

```
$logger = new Logger('main');

$logger->pushHandler(new StreamHandler(__DIR__ . '/../log/app.log',
Logger::DEBUG));
```

faire de même pour le cache :

```
$twig = new \Twig\Environment($loader, [
'cache' => '../cache',
]);
```

Et pour finir il faut appeler le fichier **conf.php** :

```
require_once("conf.php");
```

Pour se connecter à la base de donnée nous faisons un **try/catch**
On commence par déclarer la variable **$db**, puis la variable **$manager**.
Ensuite on met une log qui prévient si la connexion est réussie.
Et pour finir on lui dit qu'elle template il doit utiliser, ici le **index.html.twig**

```
try {
    $db = new PDO($dsn, $dbname, $dbpass);
    $db->setAttribute(PDO::ATTR_EMULATE_PREPARES, false); //Si toutes
les colonnes sont fausse

    $manager = new UserManager($db);

    $logger->info('Connexion réussi');

    echo $twig->render('index.html.twig',
        [
            'msg_danger' => '',
            'msg_success' => '',
            'title' => 'Liste des utilisateurs',
            'users' => $manager->getAll(),
        ]
    );
}
catch (PDOException $e) {
    print('<br/>Erreur de connexion ' . $e->getMessage());
}
```

On dit a twig que users doit prendre la variable **$manager** et appelle la méthode **getAll()**

La méthode **getAll()** est alors dans le **UserManager.php**.

```
    public function getAll(): array
    {
        $listeUser = array();
        //Retourne la liste de tous les pers
        $request = $this->_db->query('SELECT * FROM users;');
        while ($ligne = $request->fetch(PDO::FETCH_ASSOC)) // Chaque
entrée sera récupérée
        {
            $user = new User($ligne);
            $listeUser[] = $user;
        }
        return $listeUser;
    }
```

La méthode getAll() est en fait une requête pour récupérer la liste des utilisateurs.

Et pour finir le index nous allons dans le dossier *templates* et **index.html.twig**
Vous allez alors écrire votre code html en appelant **bootstrap.css**
et vous allez faire un tableau avec la syntaxe twig {% … %} en lui disant que user est égal a users.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <link rel="stylesheet" href="css/bootstrap.css">
    <title>{{title}}</title>
</head>
<body>
<h1>{{title}}</h1>

    <table class="table">
        <thead>
            <th>ID</th>
            <th>email</th>
            <th>role</th>
        </thead>
        <tbody>
```

```
    {% for user in users %}
            <tr>
            <td>{{user.id}}</td>
            <td>{{user.email}}</td>
            <td><a href="read.php?id={{user.id}}">Voir</a> <a
href="edit.php?id={{user.id}}">Modifier</a> <a
href="remove.php?id={{user.id}}">Supprimer</a></td>
            </tr>
    {% endfor %}
        </tbody>
    </table>
       <a href="add.php" class="btn btn-primary">Ajouter un
utilisateur</a>
</body>
</html>
```

Vous pouvez alors ouvrir un terminal et écrire :
PHP -S localhost:8000 -t src
Ceci va vous permettre de démarrer votre serveur web.

Ouvrez alors votre navigateur web et rendez vous sur **localhost:8000**
Si tout c'est bien passé vous aurez alors cette page :

b) Ajouter un utilisateur

Pour ajouter un utilisateur crée alors dans le dossier *templates* un fichier **add.html.twig** et dans le dossier *src* un fichier **add.php**.

Comme pour l'index on appelle ce qu'on doit utiliser. On crée des logs et le cache.

On dit a la variable **$manager** d'appeler la fonction **add()**.

```php
<?php

require_once '../vendor/autoload.php';

use Monolog\Logger;
use App\Entity\User;
use Twig\Environment;
use App\Manager\UserManager;
use Twig\Loader\FilesystemLoader;
use Monolog\Handler\StreamHandler;

$logger = new Logger('main');

$logger->pushHandler(new StreamHandler(__DIR__ . '/../log/app.log',
Logger::DEBUG));

$loader = new \Twig\Loader\FilesystemLoader('../templates');

$twig = new \Twig\Environment($loader, [
'cache' => '../cache',
]);

require_once("conf.php");

if (isset($_POST['email']) && !empty($_POST['email'])) {
    try{

        $db = new PDO($dsn, $dbname, $dbpass);
        $db->setAttribute(PDO::ATTR_EMULATE_PREPARES, false); //Si
toutes les colonnes sont fausse
        $manager = new UserManager($db);
        $user = new User(["email" => $_POST['email'], "password" =>
$_POST['password']]);

        if ($manager->add($user)) {
            $logger->info('utilisateur ajouté');
            print("Utilisateur créé<br/>");
        } else {
            print("Utilisateur NON créé<br/>");
        }

        header('Location: index.php');
```

```
    }
    catch (PDOException $e) {
        print('<br/>Erreur de connexion : ' . $e->getMessage());
    }

    }
    echo $twig->render('add.html.twig',
    [
        'msg_danger' => '',
        'msg_success' => '',
        'title' => 'Ajouter un utilisateur',
    ]
);
```

Le header(‘Location: index.php’) vous permet à chaque ajout d’utilisateur de revenir sur l’index de l’application.

La fonction **add()** se trouve alors dans le userManager :

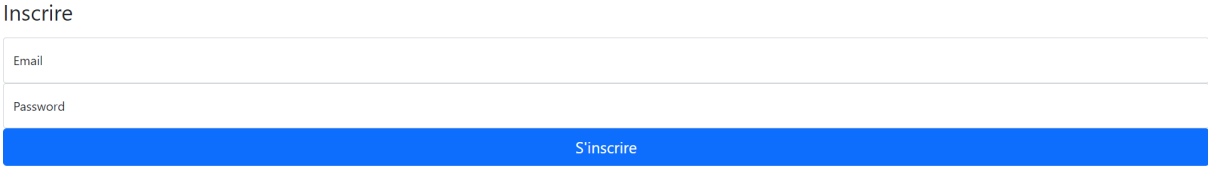
```
    public function add(User $user): bool
    {
        $result = false;
        try {
            $query = $this->_db->prepare('INSERT INTO users (`email`,
`password`) VALUES (:email, :passwd);');
            $query->bindValue(':email', $user->getEmail());
            $query->bindValue(':passwd', $user->getPassword());
            $result = $query->execute();
        } catch (Exception $e) {
            print("UNe erreur est intervenue :
'".$e->getMessage()."'");
        }
        return $result;
    }
```

Nous faisons alors une requête de type INSERT. C’est alors une requête préparée pour la sécurité du projet. Vérifiez que votre requête fonctionne. De plus, votre mot de passe sera crypté dans votre base de données.

Dans le **add.html.twig** il suffit juste de faire un simple formulaire en html :

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <link rel="stylesheet" href="css/bootstrap.css">
    <link rel="stylesheet" href="css/signin.css">
    <title>Ajouter</title>
</head>
<body>
<main class="form-signin">
    <form method="POST">
        <h1 class="h3 mb-3 fw-normal">Inscrire</h1>

        <div class="form-floating">
        <input name="email" type="email" class="form-control"
id="floatingInput" placeholder="name@example.com">
        <label for="floatingInput">Email</label>
        </div>
        <div class="form-floating">
        <input name="password" type="password" class="form-control"
id="floatingPassword" placeholder="Password">
        <label for="floatingPassword">Password</label>
        </div>


        <button class="w-100 btn btn-lg btn-primary"
type="submit">S'inscrire</button>
    </form>
</body>
</html>
```

Si cela a bien fonctionné vous aurez déjà le formulaire qui apparaît.



De plus si vous essayez d'entrer un utilisateur vous retournerez sur l'index et si tout a bien fonctionner vous aurez alors :

Si vous avez comme ci dessus c'est que tout à bien fonctionné.

c) Lire un utilisateur

Pour lire un seul utilisateur vous allez alors créer le **read.html.twig** et le **read.php**
Cela va permettre de voir les informations d'un seul utilisateur et par la suite de le modifier.

Dans le **read.php** vous allez mettre la requête getOne() tout en précisant l'id de l'utilisateur.

```
<?php
require_once '../vendor/autoload.php';

use Monolog\Logger;
use Twig\Environment;
use App\Manager\UserManager;
use Twig\Loader\FilesystemLoader;
use Monolog\Handler\StreamHandler;
$logger = new Logger('main');
$logger->pushHandler(new StreamHandler(__DIR__ . '/../log/app.log',
Logger::DEBUG));
$loader = new \Twig\Loader\FilesystemLoader('../templates');
$twig = new \Twig\Environment($loader, [
'cache' => '../cache',
]);
require_once("conf.php");
try {
    $db = new PDO($dsn, $dbname, $dbpass);
    $db->setAttribute(PDO::ATTR_EMULATE_PREPARES, false); //Si toutes
les colonnes sont fausse

    $manager = new UserManager($db);

    echo $twig->render('read.html.twig',
        [
            'msg_danger' => '',
            'msg_success' => '',
            'title' => 'Liste de l\'utilisateur',
            'user' => $manager->getOne($_GET['id']),
        ]
    );
}
catch (PDOException $e) {
```

```
    print('<br/>Erreur de connexion ' . $e->getMessage());
}
```

Dans le UserManager on crée alors le getOne en précisant l'id.

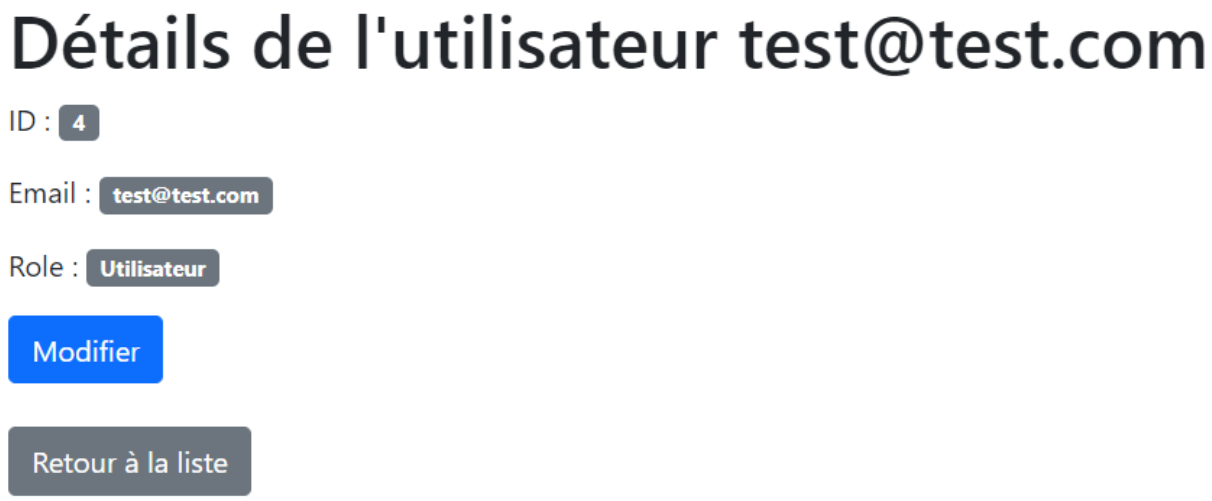
```
    public function getOne(int $id)
    {
        $sth = $this->_db->prepare('SELECT * FROM users WHERE id = ?');
        $sth-> execute(array($id));
        $ligne = $sth->fetch();
        $user = new User($ligne);
        return $user;
    }
```

Tout en faisant une requête préparé pour éviter les intrusions sur l'application WEB.

Dans le **read.html.twig** on créer alors l'interface pour avoir le détail de l'utilisateur toujours avec la syntaxe twig qui permet de ne pas mettre de code php et l'on prévoit un bouton modifier pour le edit.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <link rel="stylesheet" href="css/bootstrap.css">
    <title>{{title}}</title>
</head>
<body>
<main class="container">
    <div class="row">
        <section class="col-12">
        <h1>Détails de l'utilisateur {{user.email}}</h1>
        <p>ID : <span class="badge bg-secondary">
{{user.id}}</span></p>
        <p>Email :  <span class="badge
bg-secondary">{{user.email}}</span></p>
        <p>Role :  <span class="badge
bg-secondary">{{user.role}}</span></p>

        <p>
        <a class="btn btn-primary"
href='edit.php?id={{user.id}}'>Modifier</a><br><br>
        <a class="btn btn-secondary" href='index.php'>Retour à la
liste</a><br>
        </p>
        </section>
    </div>
</main>

</body>
</html>
```

Si tout a bien fonctionné vous allez donc avoir cette page :



Toujours grâce à la template Bootstrap.

d) Modifier un utilisateur

Pour modifier un utilisateur vous allez alors créer le **edit.html.twig** et le **edit.php**
Cela va permettre de modifier les informations d'un seul utilisateur.

Dans le **edit.php** vous allez mettre la requête update() tout en précisant l'id de l'utilisateur.

Vous pourrez seulement modifier votre Email ainsi que votre Rôle.

```php
<?php
require_once '../vendor/autoload.php';

use Monolog\Logger;
use Twig\Environment;
use App\Manager\UserManager;
use Twig\Loader\FilesystemLoader;
use Monolog\Handler\StreamHandler;

$logger = new Logger('main');

$logger->pushHandler(new StreamHandler(__DIR__ . '/../log/app.log',
Logger::DEBUG));

$loader = new \Twig\Loader\FilesystemLoader('../templates');
```

```php
$twig = new \Twig\Environment($loader, [
'cache' => '../cache',
]);

require_once("conf.php");

try {
    $db = new PDO($dsn, $dbname, $dbpass);
    $db->setAttribute(PDO::ATTR_EMULATE_PREPARES, false);
    $manager = new UserManager($db);

    if ($_POST) {
        if (
            isset($_POST['id']) && !empty($_POST['id'])
            && isset($_POST['email']) && !empty($_POST['email'])
        ) {
            $id = strip_tags($_POST['id']);
            $email = strip_tags($_POST['email']);
            $role = strip_tags($_POST['roles']);

            $user = $manager->getOne($id);
            if (!$user) {
                print($_SESSION['error'] = "Cet id n'existe pas");
                header('Location: index.php');
                exit();
            }
            $user->setEmail($email);
            $user->setRole($role);
            $manager->update($user);
            $logger->info('utilisateur modifié');
            header('Location: index.php');
        } else {
            print($_SESSION['error'] = "La formulaire est incomplet");
            header('Location: index.php');
        }
    }

    if (isset($_GET['id']) && !empty($_GET['id'])) {

        $id = strip_tags($_GET['id']);

        $user = $manager->getOne($id);
```

```
        if (!$user) {
            print($_SESSION['error'] = "Cet id n'existe pas");
            header('Location: index.php');
            exit();
        }
    } else {
        print($_SESSION['error'] = "URL invalide");
        header('Location: index.php');
        exit();
    }
} catch (PDOException $e) {
    print('<br/>Erreur de connexion : ' . $e->getMessage());
}

echo $twig->render('edit.html.twig', [
    'title' => 'Modifier utilisateur',
    'user' => $manager->getOne($_GET['id']),

]);
```

Créons directement la méthode update dans le **UserManager.php**

```
    public function update(User $user): bool
    {
        $query = $this->_db->prepare('UPDATE users SET email=:email,
role=:roles WHERE id=:id;');
        $query->bindValue(':email', $user->getEmail());
        $query->bindValue(':roles', $user->getRole());
        $query->bindValue(':id', $user->getId());
        return($query->execute());
    }
```

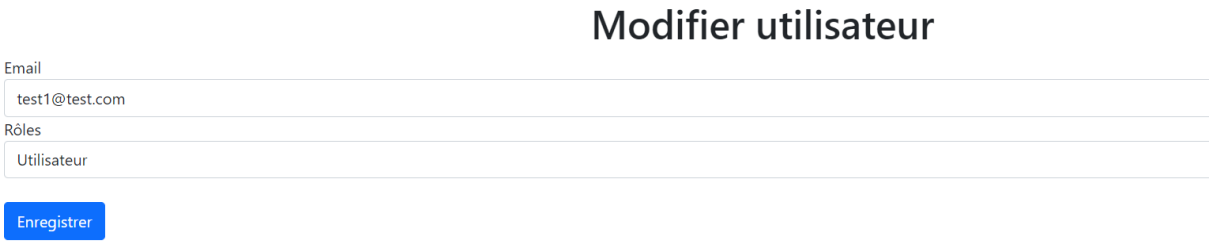
Toujours dans une requête préparée pour éviter l'intrusion. Nous utiliserons aussi une requête de type UPDATE.

Et de plus dans le **edit.html.twig** nous allons juste mettre une formulaire, avec quelque syntaxe de twig pour préremplir les champs du formulaire avec celui de l'utilisateur.

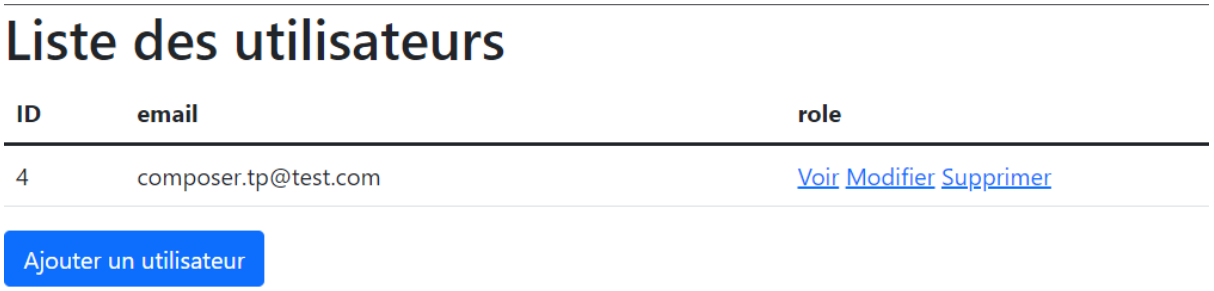
```
      <!DOCTYPE html>
      <html lang="fr">
      <head>
          <meta charset="UTF-8">
          <meta http-equiv="X-UA-Compatible" content="IE=edge">
          <meta name="viewport" content="width=device-width,
initial-scale=1.0">
          <link rel="stylesheet" href="css/bootstrap.css">
          <title>{{ title }}</title>
      </head>
      <body>
          <h1 class='text-center'>{{ title }}</h1>
                      <form method="post">
                          <div class="form-group">
                              <label for="email">Email</label>
                              <input type="text" id="email"
name="email" class="form-control" value="{{user.email}}" autofocus>
                              <label for="email">Rôles</label>
                              <input type="text" id="roles"
name="roles" class="form-control" value="{{user.role}}">
                          </div>
                          <input type="hidden" value="{{user.id}}"
name="id">
                          <br/>
                          <button class="btn
btn-primary">Enregistrer</button>
                      </form>
```

```
    </body></html>
```

Ce formulaire sert alors à modifier son email, et son rôle si vous le souhaitez.
Si vous vous souvenez, l'utilisateur avait comme email test1@test.com.
Ainsi que le rôle utilisateur.
Nous allons donc le modifier en composer.tp@test.com et rôle administrateur.



Si tout se passe bien vous serez alors redirigé vers l'index et vous verrez que votre Email à changer comme ci dessous :



L'update des utilisateurs est alors finie.

e) Supprimer un Utilisateur

Pour supprimer un utilisateur nous le ferons avec le bouton Supprimer que l'on trouve dans l'index. Nous allons donc appeler les fichiers **delete.html.twig** toujours dans le dossier *templates* et **delete.php** dans le dossier *src*.

Dans le fichier **delete.html.twig** il n'y a pas besoin de faire une page vous pouvez simplement faire le début d'une interface comme ci-dessous :

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <link rel="stylesheet" href="css/bootstrap.css">
    <title>Supprimer</title>
```

```
</head>
<body>
</body></html>
```

Néanmoins c'est différent pour le **delete.php** qui sera plus complet.
Le fichier appellera alors la méthode delete et redirigera directement vers l'index sans que vous vous en apercevrez.
Et affichera une alerte si votre utilisateur à été supprimé en vert ou si il y a une erreur une alerte s'affiche en rouge.

Pour continuer le code dans le **delete.php** ne contiendra pas grand chose si l'on enlève l'alerte :

```php
<?php

require_once '../vendor/autoload.php';

use Monolog\Logger;
use App\Entity\User;
use Twig\Environment;
use App\Manager\UserManager;
use Twig\Loader\FilesystemLoader;
use Monolog\Handler\StreamHandler;

$logger = new Logger('main');

$logger->pushHandler(new StreamHandler(__DIR__ . '/../log/app.log',
Logger::DEBUG));

$loader = new \Twig\Loader\FilesystemLoader('../templates');

$twig = new \Twig\Environment($loader, [
'cache' => '../cache',
]);

require_once("conf.php");

try {
    $db = new PDO($dsn, $dbname, $dbpass);
    $db->setAttribute(PDO::ATTR_EMULATE_PREPARES, false); //Si toutes
les colonnes sont fausse

    $manager = new UserManager($db);
    $user = $manager->getOne($_GET['id']);
```

```
    $logger->info('Supression de l\'utilisateur réussi.');

    if ($manager->delete($user)) {
        print("<div class='alert alert-success' role='alert'>
        <h4 class='alert-heading'>Utilisateut supprimé !</h4>
        <hr>
        <p class='mb-0'>Vous pouvez retourner à l'accueil du site web
via ce lien : <a href='index.php' class='alert-link'><i class='fas
fa-link'></i> index.php</a></p>
        </div>");
    } else {
        print("<div class='alert alert-danger' role='alert'>
        <h4 class='alert-heading'>Erreur!</h4>
        <p>L'utilisateur n'a pas été supprimé.</p>
        <hr>
        <p class='mb-0'>Vous pouvez retourner à l'accueil du site web
via ce lien : <a href='index.php' class='alert-link'><i class='fas
fa-link'></i> index.php</a></p>
        </div>");
    }
}
catch (PDOException $e) {
    print('<br/>Erreur de connexion ' . $e->getMessage());
}

echo $twig->render('delete.html.twig',
[
    'msg_danger' => '',
    'msg_success' => '',
    'title' => 'Supression de l\'utilisateur',
]
);
```

Comme vous le voyez il y a seulement le code principal dans le php tandis que dans la template rien ne s'y trouve.

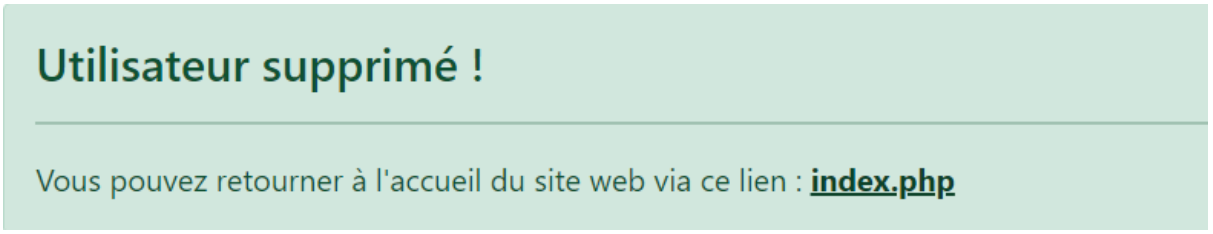
Ensuite la méthode delete est courte aussi il n'y a besoin que de 3 lignes qui sont :

```
    public function delete(User $user): bool
    {
        $query = $this->_db->prepare('DELETE FROM `users`WHERE
id=:id;');
        $query->bindValue(':id', $user->getId());
```

```
        return $query->execute();
    }
```

On fait alors une requête préparé de type Delete et on lui dit de supprimer via l'id de l'utilisateur qui est alors unique à tous le monde.

Pour finir si tout c'est bien passé et que vous avez voulu supprimer un utilisateur vous atterrissez alors sur cette petite page après avoir cliquez sur Supprimer :



Sinon vous aurez le même type d'alerte en rouge pour vous indiquer qu'il y a une erreur quelque part.

En retournant sur l'index vous verrez qu'il n'y a alors plus l'utilisateur que vous avez
choisis de supprimer.

X/ Conclusion

Comme vous avez pu le remarquez il y a des log à chaque fois qu'une action est effectué comme pour création d'un utilisateur, la modification et la suppression.

De plus si vous regardez bien vous verrez que dans votre base de données les mot de passe sont cryptés pour éviter tout problème.

Vous pouvez aussi tout retrouver sur mon Dépôt Github fonctionnel à 100% https://github.com/Isaatox/tp-composer

Il vous faudra juste installer Composer comme précisé plus haut.