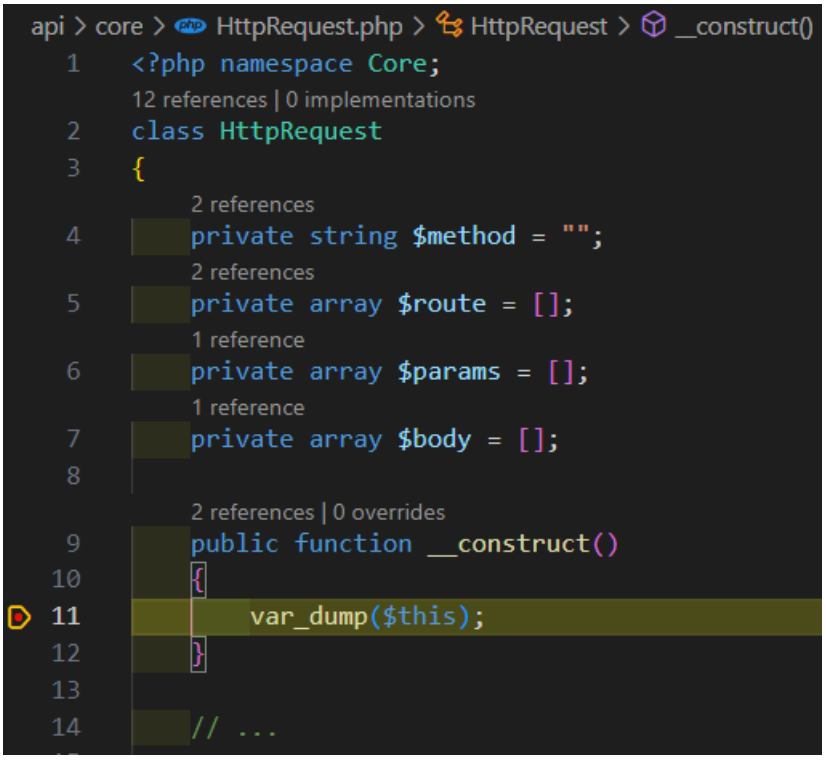
PHP Blog Project

12. API Rest : Solution pour HttpRequest - 1ère version du Router

La classe HttpRequest (une version possible)

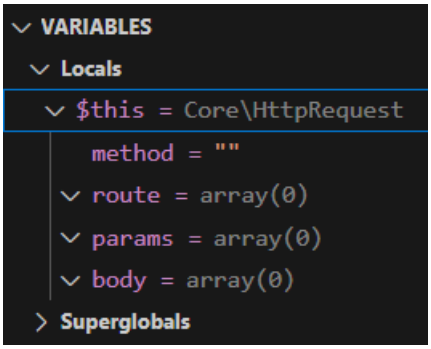
1ère étape : créer les propriétés nécessaires et le constructeur
Nous avons besoin d'une propriété privée pour chaque information nécessaire ou demandée



Nous testons en debug avec la route :
http://api.php-blog-project.loc/article/id?sort=last_published&order=desc

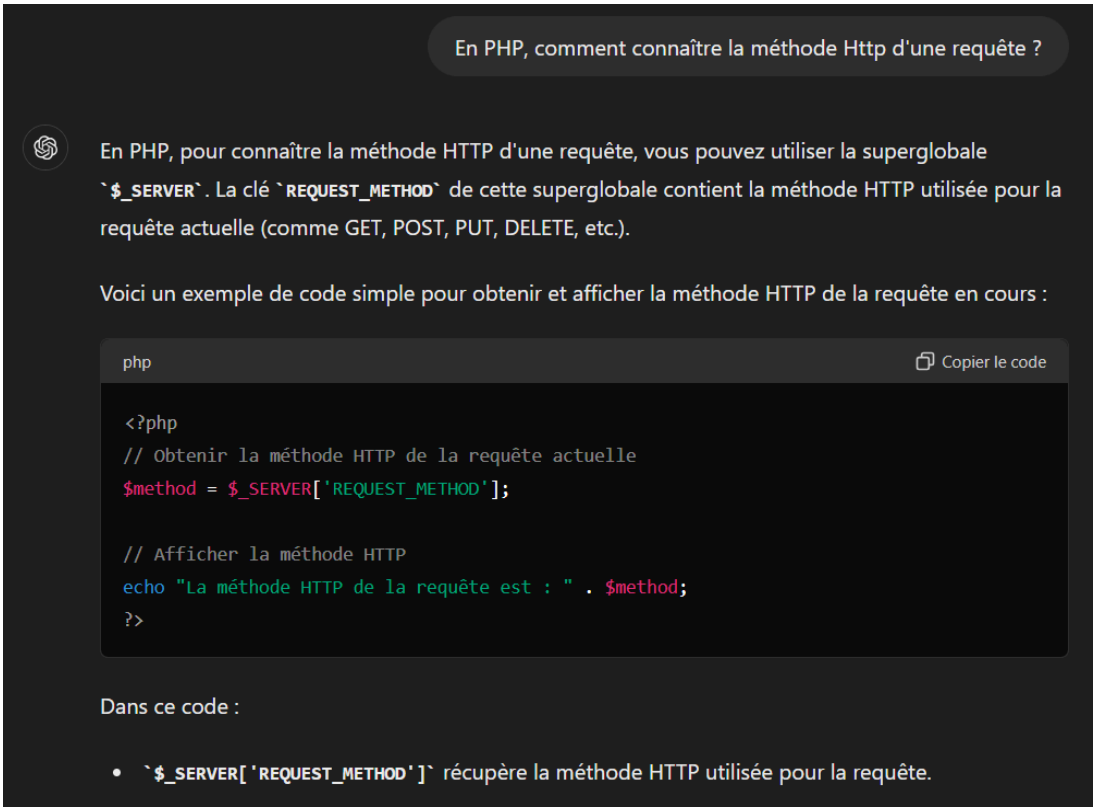
Ne pas oublier d'instancier la classe dans index.php pour tester.

Résultat

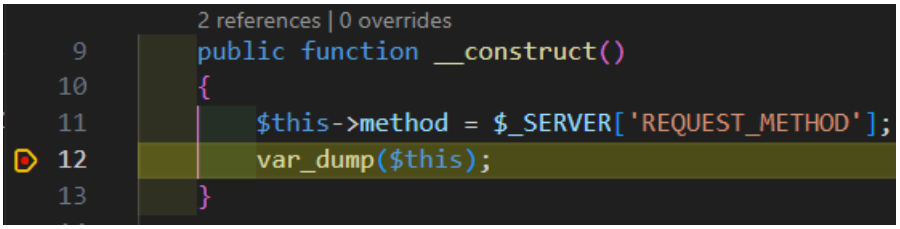


doc :
https://www.php.net/manual/fr/language.types.declarations.php

2ème étape : récupérer les infos pour les stocker dans les propriétés
Avec l'aide de ChatGPT

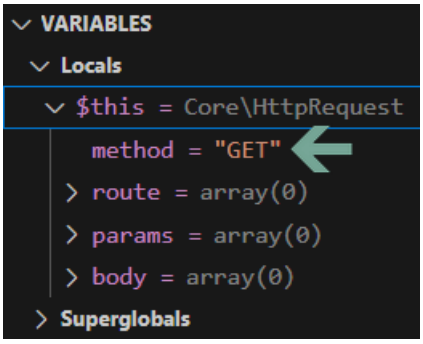


Nous complétons notre constructeur

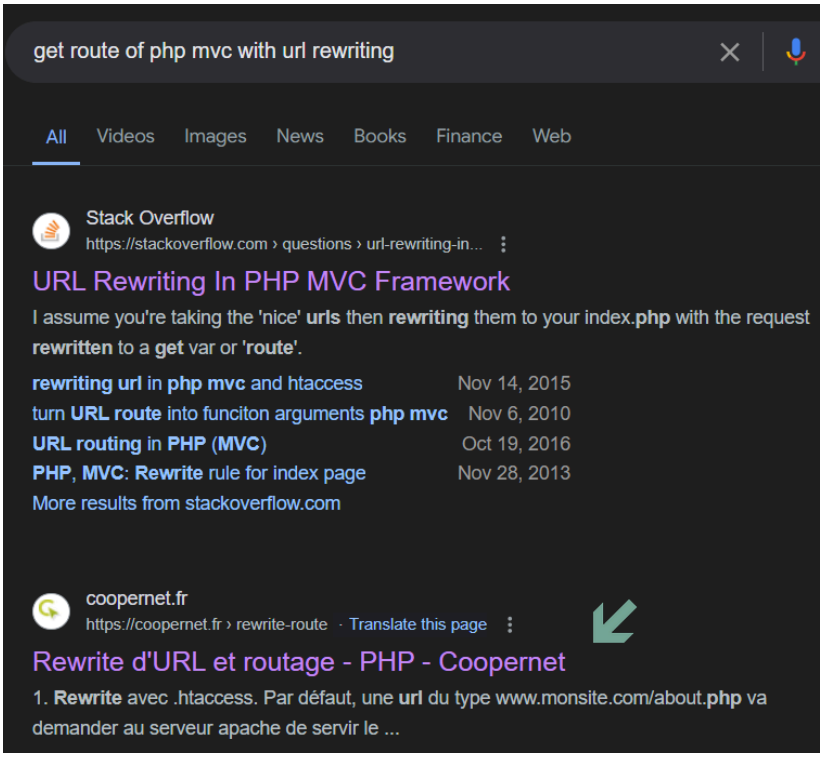


doc :
https://www.php.net/manual/en/reserved.variables.server.php

Résultat



Avec une recherche Google



2. Reconnaître le chemin demandé

Une fois que ce fichier .htaccess est en place, il va falloir tester quel est le chemin demandé et appeler le fichier php correspondant.

Exemple simpliste.

Imaginons que l'on veuille que la requête http://www.monsite.com/toto permette de servir le fichier /templates/toto.php

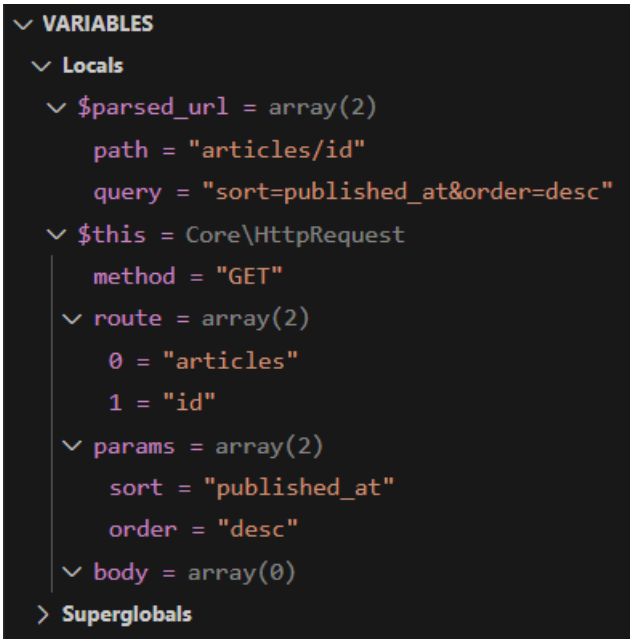
Il nous suffira de tester, dans le fichier index.php, le chemin de l'url que l'on va pouvoir récupérer avec la fonction parse_url() de la manière suivante :

```
// parse_url() analyse une URL et retourne ses composants
$parsed_url = parse_url($_SERVER['REQUEST_URI']);

// soit l'url en question a un chemin et sinon le chemin est la racine
$path = isset($parsed_url['path']) ? $parsed_url['path'] : '/';

// si le chemin est bien toto alors on fait appel au fichier /templates/toto.php
if($path == "/toto") require_once($_SERVER["DOCUMENT_ROOT"].'/templates/toto.php');
```

Nous complétons notre constructeur

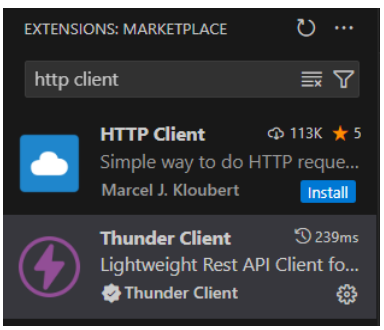
```php
2 references | 0 overrides
public function __construct()
{
    $this->method = $_SERVER['REQUEST_METHOD'];
    $parsed_url = parse_url(filter_var(trim($_SERVER['REQUEST_URI'], "/")));
    $this->route = explode('/', $parsed_url['path']);
    parse_str($parsed_url['query'] ?? "", $this->params);
    var_dump($this);
}
```

Résultat

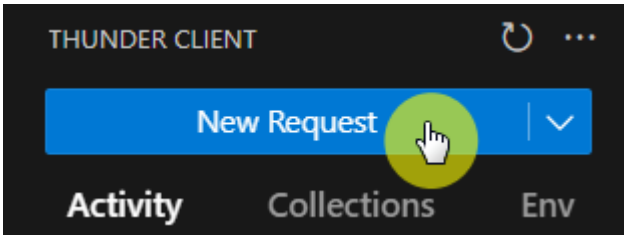


doc :
https://www.php.net/manual/en/function.parse-url
https://www.php.net/manual/fr/function.parse-str

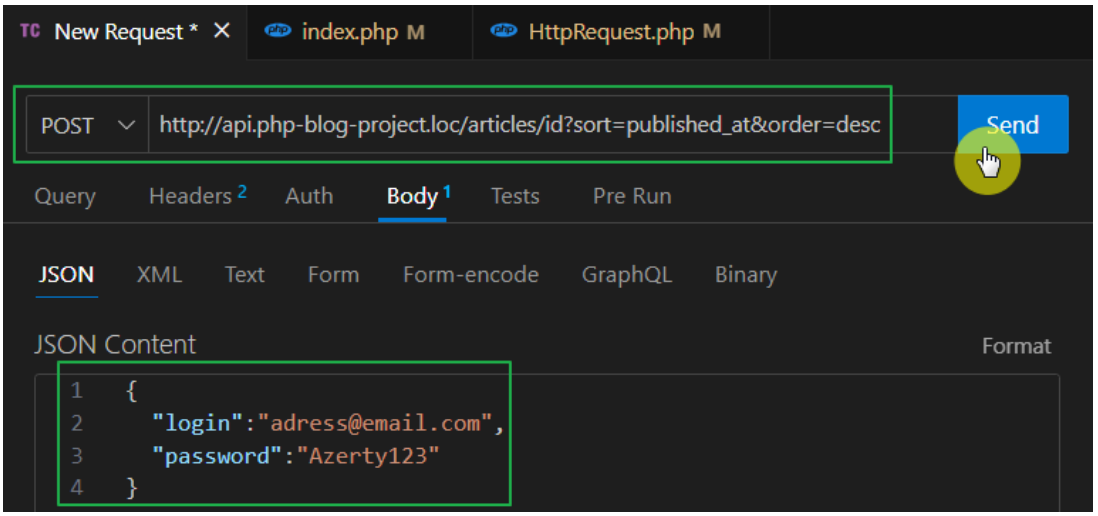
Pour le body de la requête, nous avons besoin d'exécuter des requêtes avec la méthode POST. Nous pouvons créer un formulaire et l'envoyer avec la méthode POST sur une route de notre API.
Autre solution : utiliser un client HTTP tel que Postman, ou l'extension Thunder Client de VSCode.
https://www.postman.com/
https://www.thunderclient.com/



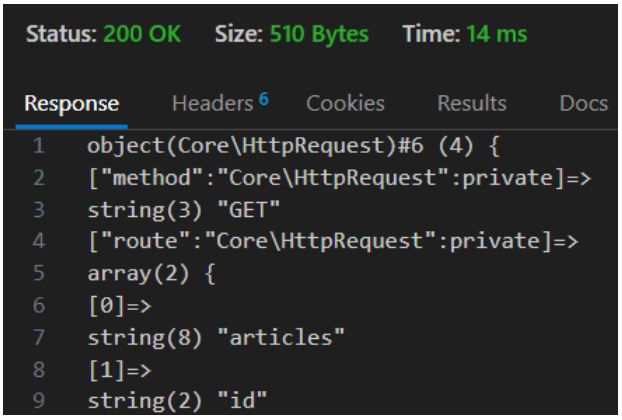
Après avoir installé l'extension Thunder Client rendez-vous sur l'onglet correspondant dans VS Code et cliquez sur New Request



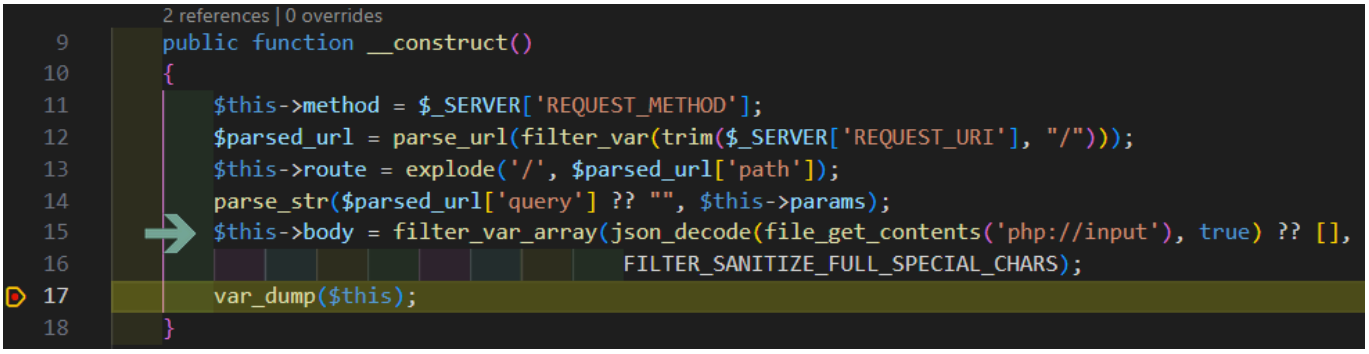
Puis paramétrez la requête et l'exécuter

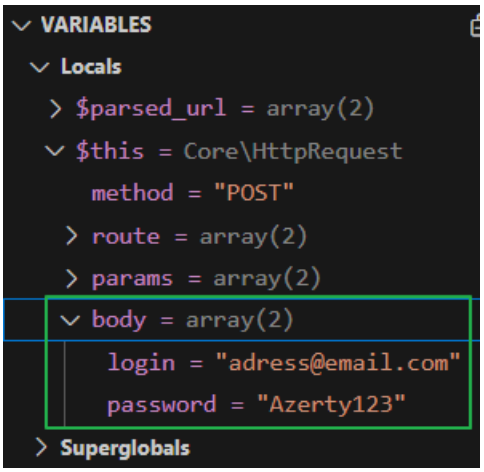


Vous obtenez la réponse qui correspond au var_dump (ligne 15 de notre constructeur)



Nous allons maintenant pouvoir essayer de récupérer le body de la requête.
Après quelques recherches … Nous complétons notre constructeur et testons à l'aide de Thunder Client





doc :
https://www.php.net/manual/fr/function.filter-var-array
https://www.php.net/manual/fr/function.json-decode
https://www.php.net/manual/fr/function.file-get-contents
https://www.php.net/manual/en/wrappers.php.php

3ème étape : transformation en singleton et méthode get d'accès aux propriétés privées

Nous allons maintenant transformer notre classe en singleton.

```
7   private array $body = [];
    4 references
8   private static $instance;
    2 references
9   private function __construct()
10  {
11      $this->method = $_SERVER['REQUEST_METHOD'];
12      $parsed_url = parse_url(filter_var(trim($_SERVER['REQUEST_URI'], "/")));
13      $this->route = explode('/', $parsed_url['path']);
14      parse_str($parsed_url['query'] ?? "", $this->params);
15      $this->body = filter_var_array(json_decode(file_get_contents('php://input'), true) ?? [],
16                                     FILTER_SANITIZE_FULL_SPECIAL_CHARS);
17      var_dump($this);
18  }
    3 references | 0 overrides
19  public static function get(){
20
21  }
```

Nous créons une variable privée pour stocker notre instance et passons le constructeur en privé.

La méthode get va permettre de récupérer l'instance du singleton, mais elle doit également permettre de récupérer l'un des attributs privés de la classe (method, route, etc …)

Nous allons donc créer une enum pour limiter les options possibles en paramètre de get.

```php
enum HttpReqAttr: string
{
    6 references
    case INSTANCE = "instance";
    1 reference
    case METHOD = "method";
    2 references
    case ROUTE = "route";
    0 references
    case PARAMS = "params";
    0 references
    case BODY = "body";
}
```

Nous ajoutons ce code dans le même fichier que la classe HttpRequest (après la classe) puisqu'il est lié à cette classe, puis nous complétons notre méthode get.

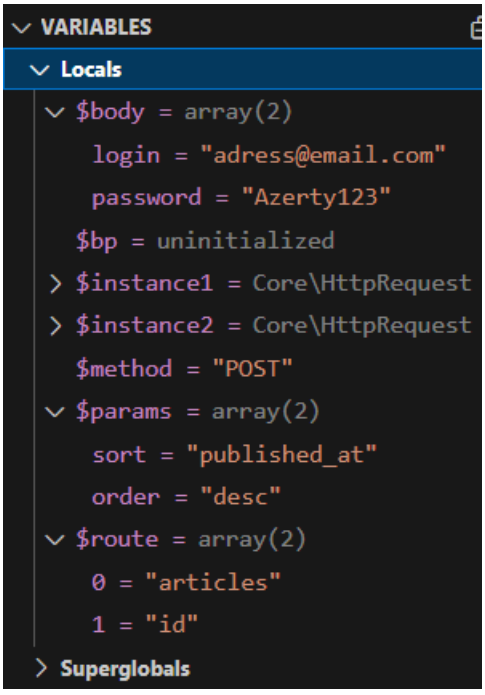
```php
    3 references | 0 overrides
    public static function get(HttpReqAttr $option = HttpReqAttr::INSTANCE)
    {
        if (is_null(self::$instance)) {
            self::$instance = new HttpRequest();
        }
        if ($option == HttpReqAttr::INSTANCE) {
            return self::$instance;
        }
        return self::$instance->{$option->value};
    }
```

Seuls option possible en paramètre : une valeur de l'enum HttpReqAttr
Si notre instance est nulle (1), nous l'initialisons avec le constructeur.
Si l'option choisie est INSTANCE (2) nous renvoyons l'instance sinon (3) nous revoyons la valeur de l'attribut correspondant à l'option (method, route, etc …)

Nous pouvons supprimer le var_dump du constructeur et retournons dans index.php pour tester notre méthode à l'aide du navigateur (en GET) ou de Thunder Client (en POST)

```php
spl_autoload_register("autoload");

$instance1 = HttpRequest::get();
$instance2 = HttpRequest::get(HttpReqAttr::INSTANCE);
$method = HttpRequest::get(HttpReqAttr::METHOD);
$route = HttpRequest::get(HttpReqAttr::ROUTE);
$params = HttpRequest::get(HttpReqAttr::PARAMS);
$body = HttpRequest::get(HttpReqAttr::BODY);
$bp = true;
```

Résultat



doc :
https://www.php.net/manual/en/language.types.enumerations.php
https://www.php.net/manual/en/language.enumerations.backed.php

Il est recommandé de typer les valeurs de retour de méthodes comme par exemple pour la méthode get qui peut renvoyer un string ($method) ou un array ($route, $params, $body) ou encore une instance de HttpRequest ($instance)

```
9 references | 0 overrides
18    public static function get(HttpReqAttr $option = HttpReqAttr::INSTANCE) : string | array | HttpRequest
19    {
```

ressources :
https://dev.to/karleb/return-types-in-php-3fip

La classe Router

Essayez de coder le routeur en vous inspirant de ce que nous avons déjà fait en MVC.
Adaptez le à la structure des routes d'une API Rest.
Dans un premier temps afficher la table, l'id (s'il est présent) et la méthode à appeler (qui dépends de la méthode HTTP utilisée pour la requête)
Servez vous de la classe HttpRequest

git :
https://github.com/DWWM-23526/PHP_BLOG_PROJECT/tree/Step12