



.HTACCESS

Se faire copain avec Apache



C'est quoi

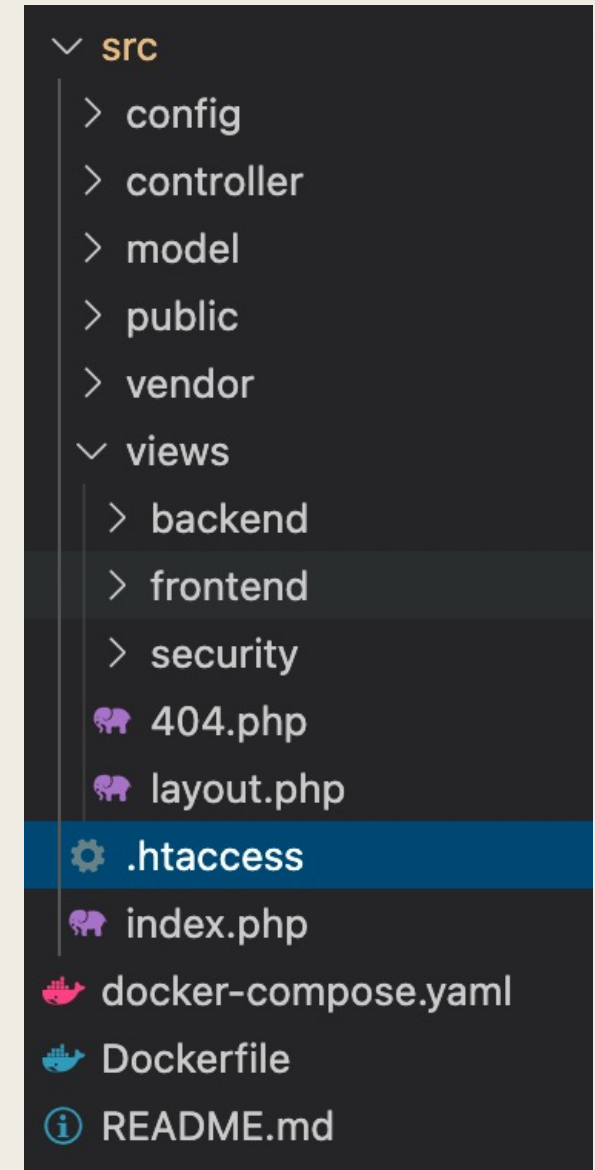
- Le fichier .htaccess est un fichier de configuration de Apache
- Sa particularité est qu'il n'est pas situé dans le répertoire de configuration Apache mais bien dans les répertoires de données du site
- Sa portée est limitée au contenu du répertoire où il réside

Ca sert à quoi ?

- Ce fichier peut faire pas mal de choses comme
 - *Personnaliser les pages d'erreurs*
 - *Rediriger des pages*
 - *Protéger un répertoire*
 - *Réécrire des URLs*

Comment ?

- On va simplement créer un fichier « .htaccess » à la racine de notre site
- La nomenclature Linux fait que ce fichier commençant par un point sera un fichier caché
- On peut y écrire des commentaires en commençant par « # »




URL REWRITING

Pour avoir de belles URL et planquer nos variables



Notre univers de travail

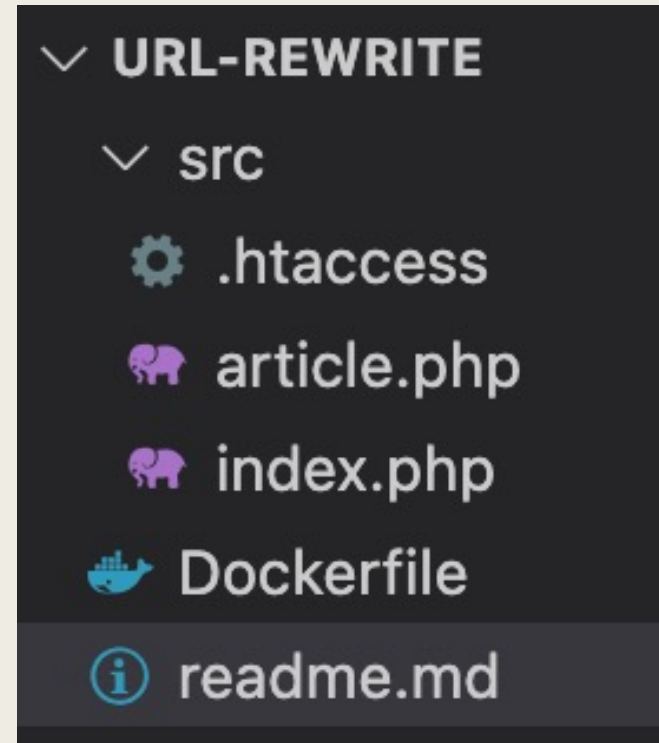
- On va commencer par se faire un petit univers de travail Docker histoire de tous avoir le même environnement
- L'image PHP / Apache de base n'a pas la modification d'URL activée par défaut, nous allons donc devoir créer un Dockerfile qui l'active
- Nous allons aussi monter un volume pour ne pas avoir à reconstruire l'image à chaque modification

```
 Dockerfile
1  FROM php:7.4-apache
2  RUN a2enmod rewrite
3
```

```
i readme.md > [abc] ## Run
1  ## Build
2
3  Il faut bien prendre le temps de build l'image puisqu'elle permet d'activer la réécriture d'URL dans Apache
4
5  ## Run
6
7  ```
8  docker run -d -p 2222:80 -v /Users/jean-francois/Desktop/url-rewrite/src:/var/www/html <BUILD NAME>
9  ```
```

Ecrire dans le .htaccess

- SPOILER ALERT : Nous allons utiliser des REGEX pour définir nos règles de réécriture
- On va commencer par faire :
 - *Un fichier index.php*
 - *Un fichier article.php*
 - *Un fichier .htaccess*
- Notre fichier index.php affichera juste un titre
- Notre fichier article.php fera un `var_dump($_GET)`



Ecrire dans le .htaccess

- On va commencer par activer le moteur de réécriture en début de fichier
- Puis nous allons ajouter une première règle, elle est composée du mot clé « RewriteRule » puis de ce que nous cherchons dans l'URL
- Cette recherche est une REGEX, le « ^ » veut dire ce que l'on cherche doit être en début de chaîne et le « \$ » que ce que l'on cherche doit être en fin de chaîne. Autrement dit, la chaîne doit être exactement égale à ce que l'on cherche
- Ensuite on spécifie ce par quoi on va le remplacer derrière le rideau (ici index.php pour retourner sur la page d'accueil)
- On va donc pouvoir taper localhost:2222/salut et retomber sur index.php

```
# Activation du moteur de réécriture  
RewriteEngine On
```

```
# Première règle de réécriture  
RewriteRule ^salut$ index.php
```


Ecrire dans le .htaccess

- Bon ok... c'est pas encore incroyable ce qu'on fait ici
- On va maintenant chercher à faire passer des parties de l'URL réécrite comme des variables récupérables par \$_GET
- On a toujours une seule règle par « RewriteRule »

```
# REGEX plus complexe :  
# on recherche "article/" suivi d'une chaîne  
# de lettres ou de chiffres et de tirets de n'importe quelle longueur  
# PUIS, séparé par un tiret, une suite de chiffres de n'importe quelle longueur  
# on va ensuite assigner la première chaîne recherchée à "$1"  
# et la seconde à "$2"  
# avant de les réinjecter dans la "vraie" URL  
  
RewriteRule ^article/([a-z0-9-]+)-([0-9]+)$ article.php?name=$1&id=$2
```

Résultat

- Ainsi, si je tape
« localhost:2222/article/je-suis-un-article-42 »
- Je vais bien récupérer :
 - *Name* = « *je-suis-un-article* »
 - *Id* = 42

```
array(2) {  
    ["name"]=>  
    string(18) "je-suis-un-article"  
    ["id"]=>  
    string(2) "42"  
}
```

Les drapeaux

- RewriteRule peut aussi prendre un 3^{ème} argument : un/des drapeau(x)
- Ces drapeaux sont des actions spéciales à effectuer
- On peut trouver parmi ces drapeaux :
 - *La redirection*
 - *Last : si une règle correspond, aucune autre ne sera appliquée après elle*
 - ...
- La liste complète est [disponible ici](#)

```
# Pour faire une redirection, on doit spécifier l'URL complète  
RewriteRule ^salut$ http://localhost:2222/article.php?name=coucou&id=45 [R=302]
```

Le bug des adresses relatives

- Je vais maintenant mettre en avant un problème avec les réécritures d'URL
- Je vais créer un fichier style.css et je vais le linker dans mes fichiers PHP
- Je vais me rendre sur
« localhost:2222/article.php?name=coucou&id=42 »...
Tout fonctionne bien
- Par contre « localhost:2222/article/coucou-42 »...
Rien ne marche plus !

```
<link rel="stylesheet" href="./css/style.css">
```

Comment corriger ça ?

- Pour le navigateur, notre réécriture « localhost:2222/article/coucou-42 » lui fait chercher un fichier style.css dans le dossier « article » il ne sait pas faire la différence entre un « vrai » chemin de fichier et une réécriture.
- Etonnement la solution se trouve en HTML avec la balise <base>
- L'élément <base> définit l'URL de base à utiliser pour recomposer toutes les URL relatives contenues dans un document
- Il ne peut y avoir qu'un seul élément <base> au sein d'un document.

```
<base href="/">  
<link rel="stylesheet" href="./css/style.css">
```

FAIRE UN ROUTEUR

Se servir de l'URL rewriting pour faire un routeur



Faire un routeur

- Dans une architecture MVC, index.php sera notre routeur et en fonction des arguments passés en URL, il exécutera des tâches différentes
- Le problème, c'est que ça donne des URL moches... utilisons .htaccess pour rendre ça plus joli

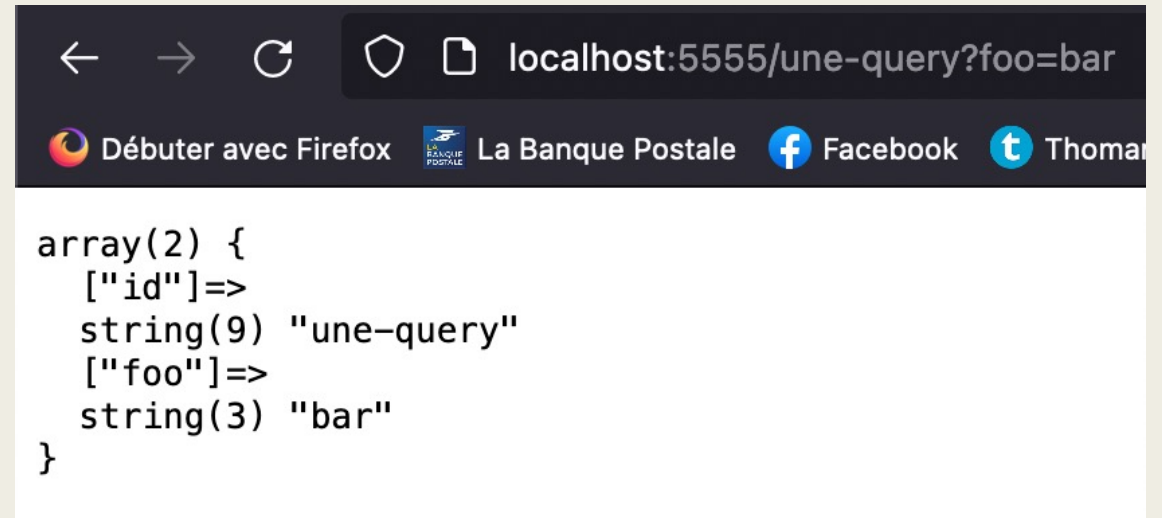
```
# Un truc un peu particulier ici, ces deux conditions veulent dire que  
# si le fichier (!-f) ou si le dossier (!-d) demandé en URL n'existe pas,  
# il faut appliquer la règle de réécriture en dessous  
# Ceci va permettre d'éviter que la réécriture soit bloquée et que tout  
# rertourne vers index.php avant même d'être traité
```

```
RewriteCond %{REQUEST_FILENAME} !-f  
RewriteCond %{REQUEST_FILENAME} !-d  
RewriteRule ^(.*)$ index.php?url=$1 [NC,L]
```

```
# Les deux drapeaux sont NoCase et Last
```

[QSA]

- Le problème maintenant, c'est que dans l'état, si on rajoute une query string dans l'URL, elle ne sera pas prise en compte, ce qui peut être problématique si on veut faire des recherches
- Le Flag QSA (Query String Append) permet de prendre en compte les requêtes en URL qui ne font pas partie de la REGEX



A screenshot of a web browser window. The address bar shows the URL `localhost:5555/une-query?foo=bar`. Below the address bar, there are several navigation links: "Débuter avec Firefox", "La Banque Postale", "Facebook", and "Thom". The main content area of the browser displays a JSON array:

```
array(2) {  
  ["id"]=>  
  string(9) "une-query"  
  ["foo"]=>  
  string(3) "bar"  
}
```



A screenshot of a code editor showing a configuration for a RewriteRule. The text is as follows:

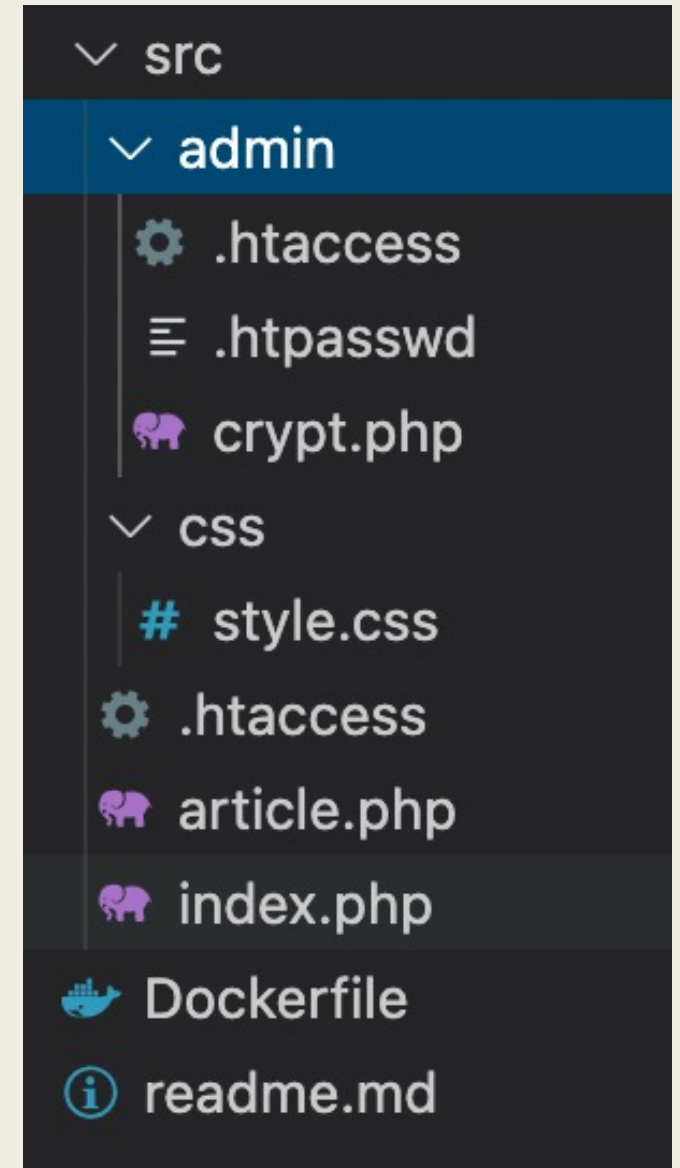
```
# Rules  
RewriteRule ^([a-z0-9-]+)$ index.php?id=$1 [NC,QSA]
```


PROTÉGER UN RÉPERTOIRE

Parce-que certaines choses ne devraient pas
être manipulées par tout le monde


.htaccess & .htpasswd

- Il est aussi possible d'utiliser le fichier .htaccess pour protéger un dossier
- Prenons l'exemple d'un dossier /admin auquel nous voulons restreindre l'accès
- Je vais créer deux fichiers dans celui-ci :
 - *.htaccess (rappelons nous que nous pouvons en avoir plusieurs)*
 - *.htpasswd qui va contenir les clés login / password valides*



.htaccess

- Notre .htaccess va contenir au moins 4 instructions :
 - *AuthName* : Le texte affiché à l'utilisateur pour qu'il se connecte
 - *AuthType* : Le type d'autorisation à utiliser, la plus commune est « Basic » avec un simple login et mot de passe
 - *AuthUserFile* : C'est le chemin d'accès absolu vers .htpasswd
 - *Require* : Ce qu'il faut pour pouvoir accéder, le plus simple est « valid-user »

```
src > admin >  .htaccess
1  AuthName "Page d'administration protégée"
2  AuthType Basic
3  AuthUserFile "/var/www/html/admin/.htpasswd"
4  Require valid-user
```

Realpath()

- On nous demande donc le chemin d'accès absolu, ça dépend pas mal du serveur et ça peut être délicat à trouver.
- Dans le cas d'un conteneur Docker Apache, c'est assez simple : « /var/www/html »
- En cas de doute, on peut « tricher » et utiliser la fonction PHP realpath()
- Je crée un fichier temporaire là où je vais mettre mon .htpasswd et je lui demande de m'afficher le realpath()

```
<?= realpath(__DIR__); ?>
```

.htpasswd

- C'est le fichier qui va contenir toutes les clés login / password
- Attention [les password doivent être encryptés](#)
- Le plus simple pour encrypter un password est encore une fois d'utiliser une fonction PHP : `crypt()`, elle prend deux arguments dont un est optionnel :
 - *Le password à encrypter*
 - [Le salt](#) (*n'est plus optionnel à partir de PHP 8*)

```
src > admin > ≡ .htpasswd
1   admin:$1$tHdobdmN$NkKJ0kexW4hb10i.3Chvh0
2
```