

L'environnement de développement

Objectif : décrire l'installation et la configuration des outils de développement d'une plateforme de covoiturage intégrant une approche écologique.

Préambule : je travaille actuellement avec Windows 11. La présente documentation détaille les installations sur ce système d'exploitation, cependant tous les outils et technologies choisis sont compatibles avec Linux et macOS (à l'exception de la console et du gestionnaire de paquets Windows).

Pour ce projet, j'aurai besoin de :

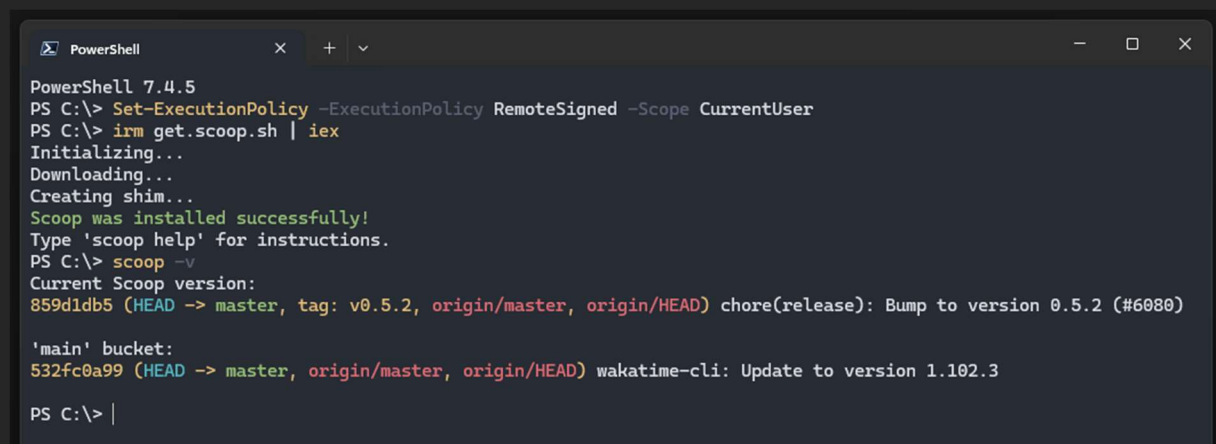
1. **Terminal et Scoop** - page 2 -
2. **Git, Git Bash, GitFlow et GitHub** - pages 3 à 5 -
3. **PHP, sa configuration et Composer** - pages 5 à 8 -
4. **Node.js et npm** - page 9 -
5. **Docker et Docker-Compose** - page 10 -
6. **VS Code et ses extensions** - page 11 -
7. **Symfony et la configuration du projet** - pages 12 à 17 -
8. **Bases de données** - pages 18 et 19 -
9. **Outils annexes** - page 20 -

1. Terminal et Scoop

Depuis qu'elle s'est améliorée, je réutilise la console Windows. Elle est préinstallée, s'appelle **Terminal** et se met à jour via le Microsoft Store. On peut maintenant :

- Travailler simultanément avec plusieurs sessions grâce à la gestion par onglets.
- Accéder, en plus des classiques Invite de commande et Windows PowerShell, à PowerShell : plus léger et plus rapide que son prédécesseur, il est multi-plateforme (actuellement en version 7).
- Bénéficier de la prise en charge intégrée de WSL2 pour utiliser directement des distributions Linux dans la console.
- Accessoirement, utiliser Azure Cloud Shell pour y gérer des ressources Azure.

Scoop est un gestionnaire de paquets Windows - pour l'installer :



```
PowerShell 7.4.5
PS C:\> Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
PS C:\> irm get.scoop.sh | iex
Initializing...
Downloading...
Creating shim...
Scoop was installed successfully!
Type 'scoop help' for instructions.
PS C:\> scoop -v
Current Scoop version:
859d1db5 (HEAD -> master, tag: v0.5.2, origin/master, origin/HEAD) chore(release): Bump to version 0.5.2 (#6080)

'main' bucket:
532fc0a99 (HEAD -> master, origin/master, origin/HEAD) wakatime-cli: Update to version 1.102.3

PS C:\> |
```

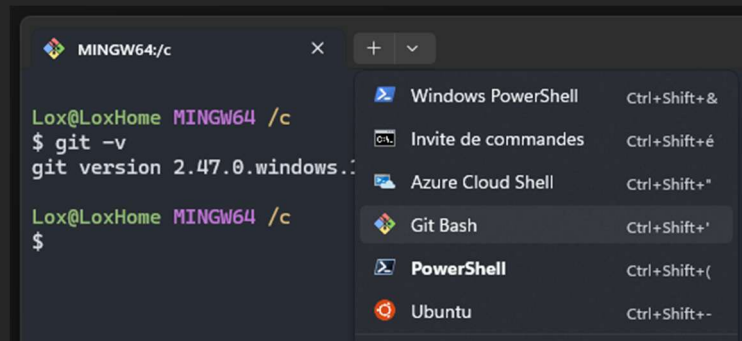
- Pour mettre à jour scoop lui-même : `scoop update`.
- Pour mettre à jour tous les paquets installés : `scoop update --all`.
- Pour mettre à jour un paquet précis : `scoop update <nom_du_paquet>`.
- **Documentation** : <https://scoop.sh/> et <https://github.com/ScoopInstaller/Scoop/wiki>.

2. Git, Git Bash, GitFlow et GitHub

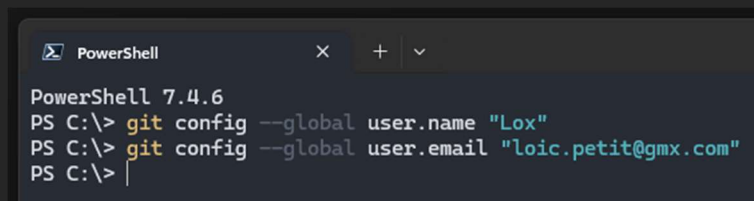
a. Git, Git Bash et GitFlow

On peut les installer par la commande `scoop install git`, mais je préfère utiliser l'exécutable disponible sur <https://git-scm.com/downloads/win> pour gérer plus finement leur intégration dans Windows (notamment dans les menus contextuels).

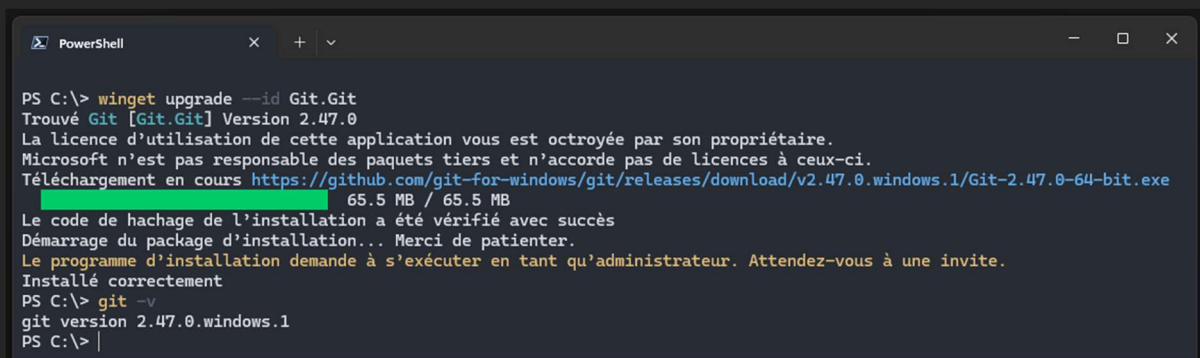
Après cette étape, les commandes `git` et `git flow` sont accessibles dans toutes les consoles, on peut aussi ouvrir une console Git Bash depuis le Terminal :



Configuration de l'identité Git (globalement) :



Pour mettre Git à jour s'il est déjà installé :



Documentation : <https://git-scm.com/doc>.

b. GitHub

Si ce n'est pas déjà fait, il faut se créer un compte : <https://github.com/signup>.

Une fois inscrit, il est préférable d'activer la double authentification (depuis le menu compte, « Settings > Password and authentication »).

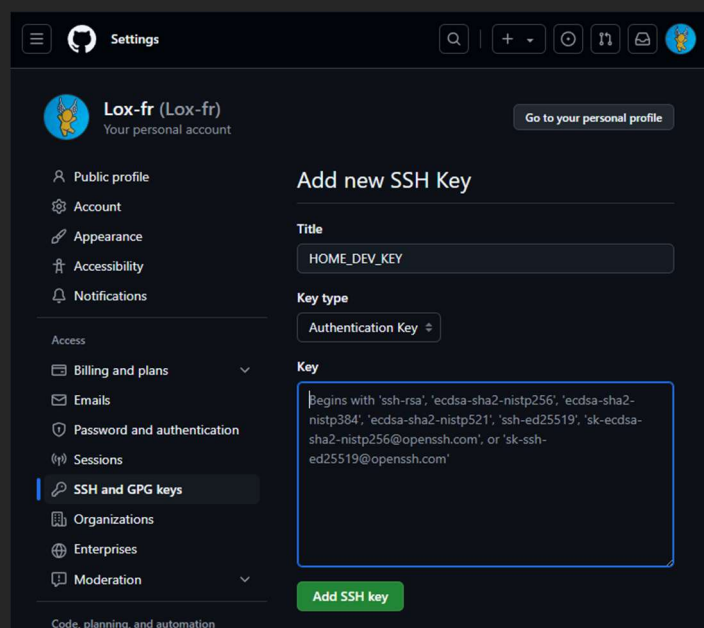
Il faut maintenant lier son compte GitHub à son environnement local. Pour cela, l'utilisation d'une connexion SSH pour l'authentification est nécessaire. OpenSSH est présent par défaut dans Windows (%SystemRoot%\System32\OpenSSH\).

Il est recommandé de configurer une clé SSH pour éviter d'avoir à saisir son mot de passe à chaque interaction. Il est préférable d'utiliser une clé Ed25519 plutôt que RSA et de renseigner une passphrase quand elle est demandée.

Pour générer la paire de clés SSH :

```
PowerShell 7.4.6
PS C:\> ssh-keygen -t ed25519 -C "loic.petit@gmx.com"
Generating public/private ed25519 key pair.
Enter file in which to save the key (C:\Users\Lox/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in C:\Users\Lox/.ssh/id_ed25519
Your public key has been saved in C:\Users\Lox/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:6vE52jEChul/TxcnjH3z0CKeLf3mXbach9mf1JInrqI loic.petit@gmx.com
The key's randomart image is:
+--[ED25519 256]--+
|
|   o   +
|  oo  S = +
| . . . . B + . .
| . . + = + ==
| . . B.O.. ***B|
| . . ooEo . . +==
+-----[SHA256]-----+
PS C:\> |
```

Enfin, il faut ajouter la clé publique générée dans son compte GitHub en copiant le contenu (texte) du fichier « id_ed25519.pub » dans « Settings > SSH and GPG keys > New SSH key » (depuis le menu compte).



Pour vérifier la connexion :

```
PowerShell
PowerShell 7.4.6
PS C:\> ssh -T git@github.com
Hi Lox-fr! You've successfully authenticated, but GitHub does not provide shell access.
PS C:\> |
```

Documentation : <https://docs.github.com/fr>.

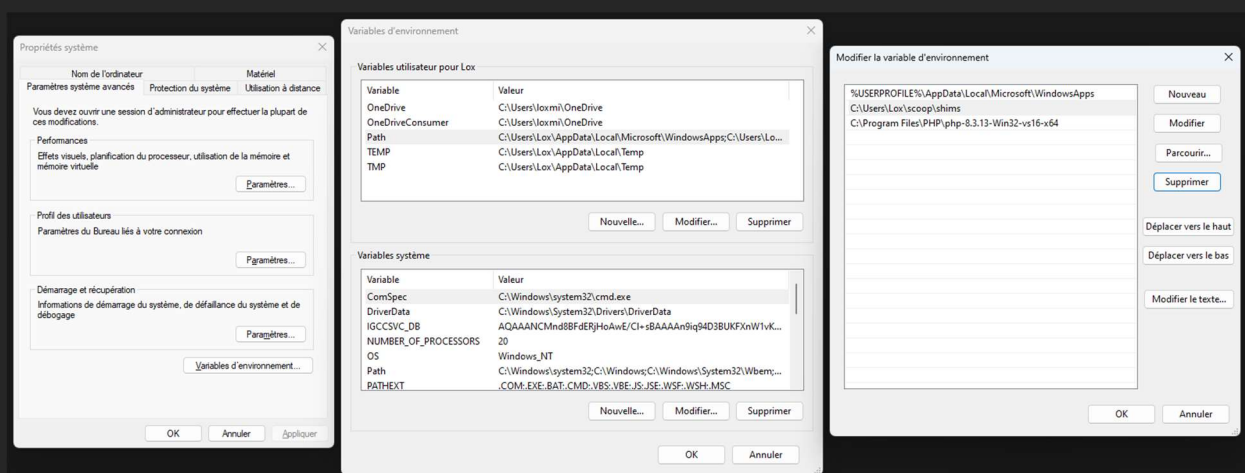
3. PHP, sa configuration et Composer

a. PHP

Il est possible d'installer PHP via Scoop avec les commandes `scoop install php` ou `scoop install php-nts`.

Sinon, on peut l'installer manuellement : il faut télécharger les fichiers binaires de la version souhaitée depuis le site officiel de PHP (<https://www.php.net/downloads.php>). Je choisis la version 8.3.13, 64 bits Thread Safe au format Zip et l'extrait dans le dossier C:\Program Files\PHP\ php-8.3.13-Win32-vs16-x64. À l'intérieur de ce dossier, je renomme le fichier « php.ini-development » en « php.ini ».

Si on l'a installé manuellement, il faut configurer la variable d'environnement : dans Windows, cela se fait depuis « Paramètres > Système > Informations système > Paramètres avancés du système », puis dans l'onglet « Paramètres systèmes avancés », et « Variables d'environnement ... ». Dans la fenêtre qui s'ouvre on clique sur la ligne « path » des « Variables d'utilisateurs » et on y ajoute une ligne à celles déjà présentes : « C:\Program Files\PHP\php-8.3.13-Win32-vs16-x64 ».



Documentation : <https://www.php.net/docs.php>.

b. Sa configuration

Elle se fait dans le fichier php.ini vu précédemment.

Pour commencer, il faut décommenter (c'est-à-dire enlever le point-virgule) la ligne « realpath_cache_size » et mettre sa valeur à « 5M », cela augmentera les performances :

```
356 ; Determines the size of the realpath cache to be used by PHP. This value should
357 ; be increased on systems where PHP opens many files to reflect the quantity of
358 ; the file operations performed.
359 ; Note: if open_basedir is set, the cache is disabled
360 ; https://php.net/realpath-cache-size
361 realpath_cache_size = 5M
```

Ensuite il faut renseigner l'endroit où sont stockées les extensions (pour pouvoir les activer) en décommentant la ligne « extension_dir = "ext" » :

```
774 ; Directory in which the loadable extensions (modules) reside.
775 ; https://php.net/extension-dir
776 ;extension_dir = "."
777 ; On windows:
778 extension_dir = "ext"
```

Pour répondre à des besoins génériques, j'active les extensions curl, fileinfo, gd, intl, mbstring, openssl, pdo_mysql, pdo_pgsql, zip, opcache :

```
936
937 extension=curl
938 ;extension=ffi
939 ;extension=ftp
940 extension=fileinfo
941 extension=gd
942 ;extension=gettext
943 ;extension=gmp
944 extension=intl
945 ;extension=imap
946 extension=mbstring
947 ;extension=exif ; Must be after mbstring as it depends on it
948 ;extension=mysqli
949 ;extension=oci8_12c ; Use with Oracle Database 12c Instant Client
950 ;extension=oci8_19 ; Use with Oracle Database 19 Instant Client
951 ;extension=odbc
952 extension=openssl
953 ;extension=pdo_firebird
954 extension=pdo_mysql
955 ;extension=pdo_oci
956 ;extension=pdo_odbc
957 extension=pdo_pgsql
958 ;extension=pdo_sqlite
959 ;extension=pgsql
960 ;extension=shmop
961
962 ; The MIBS data available in the PHP distribution must be installed.
963 ; See https://www.php.net/manual/en/snmp.installation.php
964 ;extension=snmp
965
966 ;extension=soap
967 ;extension=sockets
968 ;extension=sodium
969 ;extension=sqlite3
970 ;extension=tidy
971 ;extension=xsl
972 extension=zip
973
974 zend_extension=opcache
975
```

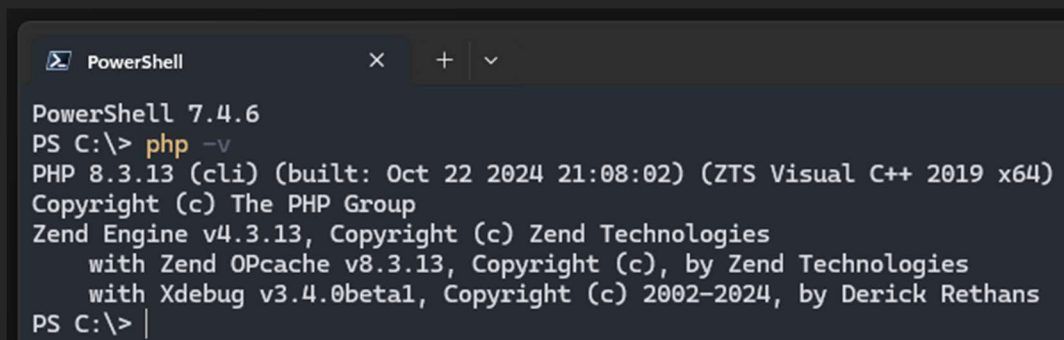
Il y a également des extensions qui ne sont pas livrées par défaut avec PHP et que je souhaite quand même installer : Xdebug pour améliorer les rapports de tests, APCu qui améliore les temps de réponse et MongoDB, nécessaire pour la gestion des bases de données NoSQL MongoDB.

Pour faire cela, je commence par récupérer les fichiers dll compatibles (disponibles sur <https://xdebug.org/download>, <https://pecl.php.net/package/APCu/5.1.24/windows> et <https://pecl.php.net/package/mongodb/1.20.0/windows>), et les place dans le fichier « ext » de PHP.

Puis je les configure dans php.ini comme suit :

```
976 [xdebug]
977 zend_extension=xdebug-3.4.0beta1-8.3-vs16-x86_64
978 xdebug.mode=coverage
979 xdebug.start_with_request=yes
980 xdebug.client_host=127.0.0.1
981 xdebug.client_port=9003
982 xdebug.log="C:\Projets\xdebug.log"
983
984 [apcu]
985 extension=apcu
986 apc.enabled=1
987 apc.enable_cli=1
988
989 [mongodb]
990 extension=mongodb
```

Pour vérifier que tout a fonctionné, la commande `php -v` doit donner :



```
PowerShell 7.4.6
PS C:\> php -v
PHP 8.3.13 (cli) (built: Oct 22 2024 21:08:02) (ZTS Visual C++ 2019 x64)
Copyright (c) The PHP Group
Zend Engine v4.3.13, Copyright (c) Zend Technologies
    with Zend OPcache v8.3.13, Copyright (c), by Zend Technologies
    with Xdebug v3.4.0beta1, Copyright (c) 2002-2024, by Derick Rethans
PS C:\> |
```

En complément, la commande `php -m` donne les extensions activées.

Documentations : <https://xdebug.org/docs/>, <https://www.php.net/manual/fr/book.apcu.php> et <https://www.mongodb.com/docs/drivers/php-drivers/>.

c. Composer

C'est le gestionnaire de paquets PHP choisi, il peut s'installer avec la commande `scoop install composer`:

```
PowerShell
PS C:\> scoop install composer
Updating Scoop...
Updating Buckets...
* 990093c0c574 x264: Update to version 3198      main
* c7cd453f99e5 tinygo: Update to version 0.34.0  main
* f5abf0056cad oxlint: Update to version 0.10.3  main
* 5724feb7ab75 oha: Update to version 1.4.7     main
* 14c7e739ec6c ntop: Update to version 0.3.20    main
Scoop was updated successfully!
Installing 'composer' (2.8.1) [64bit] from 'main' bucket
Loading composer.phar from cache
Checking hash of composer.phar ... ok.
Running pre_install script...done.
Linking ~\scoop\apps\composer\current => ~\scoop\apps\composer\2.8.1
Creating shim for 'composer'.
Adding ~\scoop\apps\composer\current\home\vendor\bin to your path.
Persisting home
'composer' (2.8.1) was installed successfully!
Notes
-----
'composer selfupdate' is aliased to 'scoop update composer'
'composer' suggests installing 'php' or 'php-nts'.
PS C:\> composer -V
Composer version 2.8.1 2024-10-04 11:31:01
PHP version 8.3.13 (C:\Program Files\PHP\php-8.3.13-Win32-vs16-x64\php.exe)
Run the "diagnose" command to get more detailed diagnostics output.
PS C:\> |
```

La commande `composer -V` permet de vérifier que la CLI fonctionne en donnant la version de composer installée.

Documentation : <https://getcomposer.org/doc/>.

4. Node.js et npm

Node.js est l'environnement d'exécution pour Javascript côté serveur et npm le gestionnaire de paquets par défaut de Node.js. On peut installer les deux en même temps avec la commande `scoop install nodejs` :

```
PowerShell 7.4.6
PS C:\> scoop install nodejs
Installing 'nodejs' (23.1.0) [64bit] from 'main' bucket
Loading node-v23.1.0-win-x64.7z from cache
Checking hash of node-v23.1.0-win-x64.7z ... ok.
Extracting node-v23.1.0-win-x64.7z ... done.
Linking ~\scoop\apps\nodejs\current => ~\scoop\apps\nodejs\23.1.0
Adding ~\scoop\apps\nodejs\current\bin to your path.
Adding ~\scoop\apps\nodejs\current to your path.
Persisting bin
Persisting cache
Running post_install script...done.
'nodejs' (23.1.0) was installed successfully!
PS C:\> node -v
v23.1.0
PS C:\> npm -v
10.9.0
PS C:\> |
```

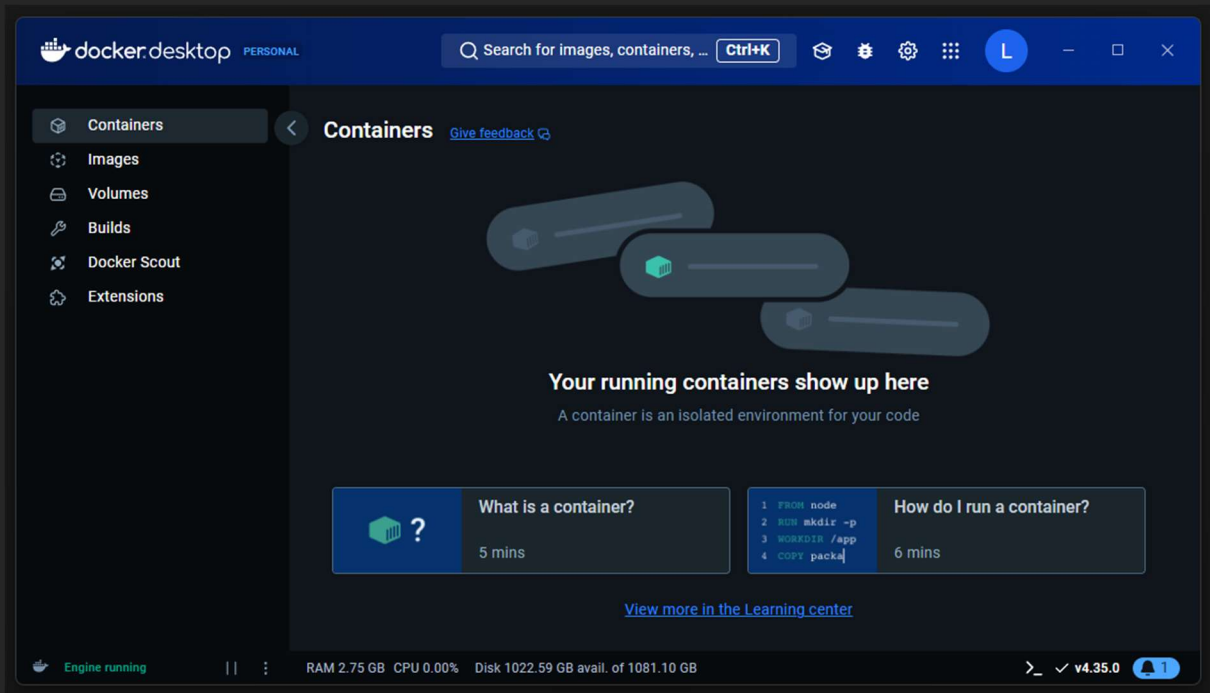
Les commandes `node -v` et `npm -v` permettent de vérifier que les deux CLI fonctionnent bien.

Documentations : <https://nodejs.org/docs/latest/api/> et <https://docs.npmjs.com/>.

5. Docker et Docker-Compose

Il est possible d'installer Docker et Docker-Compose avec les commandes Scoop, mais pour mieux gérer l'intégration dans Windows, j'utilise l'exécutable disponible sur <https://www.docker.com/> et suit les consignes du Docker Desktop Installer fourni.

Cela installe Docker Desktop, une interface graphique facile à prendre en main, ainsi que les CLI de Docker et Docker-Compose.



Vérification du bon fonctionnement des CLI :

```
PowerShell
PowerShell 7.4.6
PS C:\> docker -v
Docker version 27.3.1, build ce12230
PS C:\> docker-compose -v
Docker Compose version v2.29.2-desktop.2
PS C:\> |
```

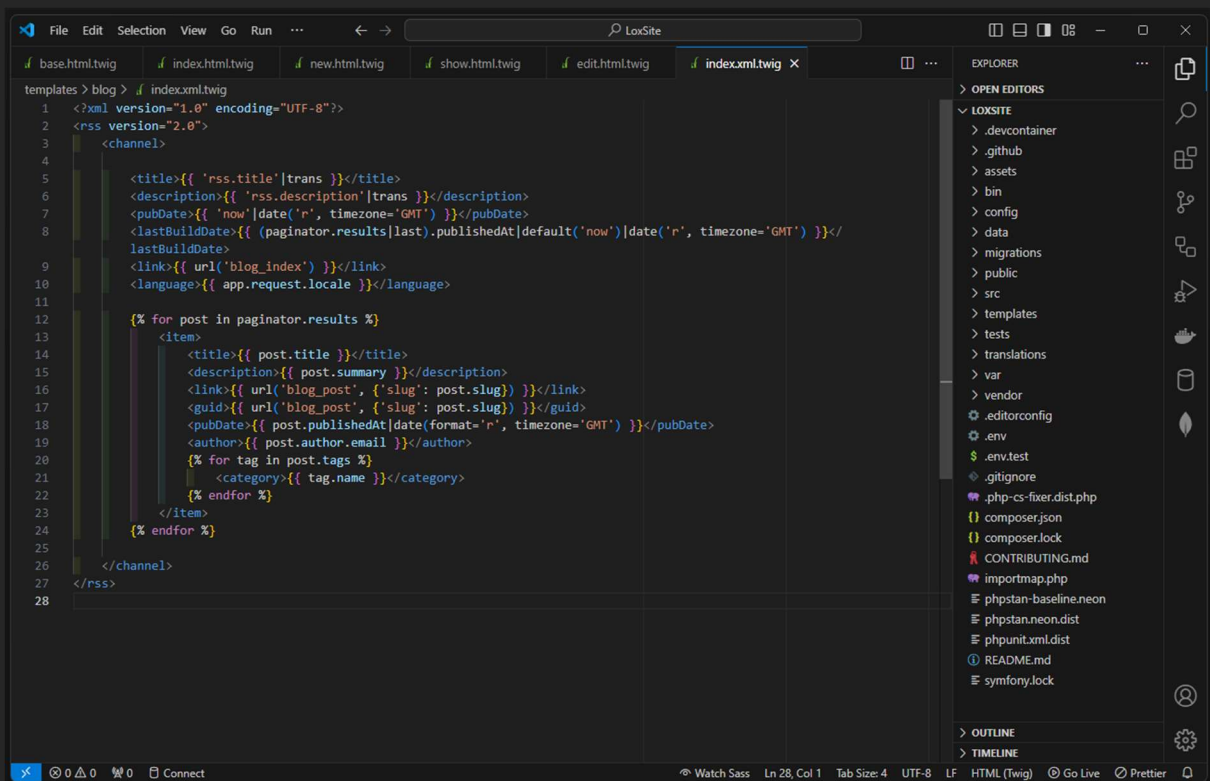
Documentation : <https://docs.docker.com/>.

6. VS Code et ses extensions

Bien que ce ne soit pas un IDE, j'opte pour cet éditeur de code gratuit par habitude. Ses nombreuses extensions disponibles le rendent très polyvalent et personnalisable.

Là encore, il est facilement installable par Scoop, mais pour mieux l'intégrer dans Windows (notamment dans les menus contextuels), je préfère utiliser l'installateur fourni sur <https://code.visualstudio.com/>.

Je ne détaillerai pas ici toutes les extensions que j'ai installées (ces choix sont en partie subjectifs), mais parmi celles-ci, certaines me semblent importantes : Composer, PHP Intelephense, PHP Namespace Resolver, PHP Debug, PHP DocBlocker, php cs fixer, SQLTools, MongoDB for VS Code, ESLint, Docker, Prettier et indent-rainbow.



L'extension PHP-CS-Fixer est préconfigurée par ces quelques lignes dans settings.json :

```
{
  "php-cs-fixer.executablePath": "${extensionPath}/php-cs-fixer.phar",
  "php-cs-fixer.config": ".php-cs-fixer.dist.php",
  "php-cs-fixer.allowRisky": true,
  "[php]": {
    "editor.defaultFormatter": "junstyle.php-cs-fixer"
  },
}
```

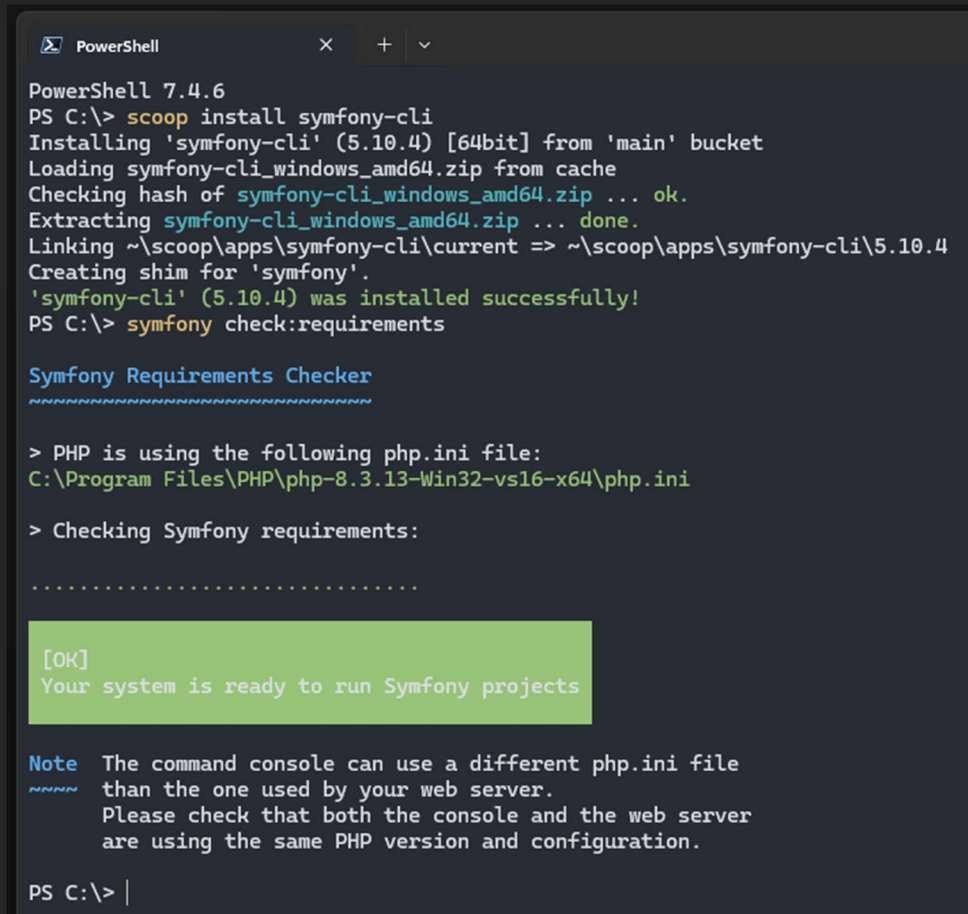
Documentation : <https://code.visualstudio.com/docs>.

7. Symfony et la configuration du projet

a. Installation de Symfony CLI

Avec la commande `scoop install symfony-cli`.

Vérification de l'installation avec la commande `symfony check:requirements` qui vérifie si mon ordinateur répond aux exigences requises par Symfony :



```
PowerShell 7.4.6
PS C:\> scoop install symfony-cli
Installing 'symfony-cli' (5.10.4) [64bit] from 'main' bucket
Loading symfony-cli_windows_amd64.zip from cache
Checking hash of symfony-cli_windows_amd64.zip ... ok.
Extracting symfony-cli_windows_amd64.zip ... done.
Linking ~\scoop\apps\symfony-cli\current => ~\scoop\apps\symfony-cli\5.10.4
Creating shim for 'symfony'.
'symfony-cli' (5.10.4) was installed successfully!
PS C:\> symfony check:requirements

Symfony Requirements Checker
#####

> PHP is using the following php.ini file:
C:\Program Files\PHP\php-8.3.13-Win32-vs16-x64\php.ini

> Checking Symfony requirements:

.....

[OK]
Your system is ready to run Symfony projects

Note The command console can use a different php.ini file
than the one used by your web server.
Please check that both the console and the web server
are using the same PHP version and configuration.

PS C:\> |
```

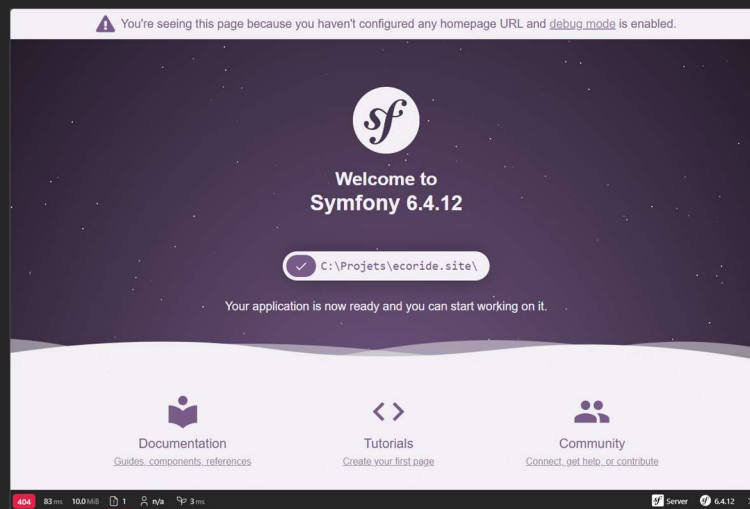
b. Initialisation du projet

Après m'être placé dans le dossier où je souhaite stocker localement le projet, j'exécute la commande `symfony new ecoride.site --version=lts --webapp` qui installe les fichiers de Symfony 6.4 dans un dossier ecoride.site automatiquement créé.

En me plaçant à l'intérieur (`cd ecoride.site`), j'exécute `symfony serve -d` qui lance le serveur local de Symfony puis `symfony open:local` qui ouvre le projet dans le navigateur par défaut.

Pour commencer à travailler, on exécute `code .` qui ouvre le projet dans l'éditeur de code par défaut (VS Code pour moi).

La première page que l'on voit sur <https://127.0.0.1:8000/> est la suivante :



Documentation : <https://symfony.com/doc/6.4/setup.html>.

c. Hébergement du code sur GitHub

Après m'être connecté à mon compte GitHub, je crée un nouveau repository (« ecoride.site »). Depuis la console, j'exécute

```
git remote add origin git@github.com:Lox-fr/ecoride.site.git.
```

Le projet est maintenant connecté à GitHub, je peux faire un

```
git push -u origin main
```

 et vérifier le bon fonctionnement sur GitHub.

d. Installation et configuration de dépendances PHP

Je commence par organiser le développement avec `git flow init` (je laisse les noms de branches par défaut) : je suis maintenant sur la branche « develop » que je publie directement avec la commande `git push -u origin develop`.

Je crée une branche spécifique pour la configuration du projet

```
git flow feature start initial-configuration.
```

PHPUnit est installé avec Symfony par défaut (si l'option `-webapp` a été utilisée), il est configuré par le fichier « `phpunit.xml.dist` » (que je laisse tel quel). Je m'assure juste qu'il est opérationnel en exécutant `php bin/phpunit` : aucun test n'est exécuté car il n'y en a pas encore mais je m'assure que la commande ne renvoie aucune erreur. Pour avoir plus de détail, on peut ajouter l'option `--testdox` à cette commande. Grâce à la présence de Xdebug, on peut aussi générer un rapport de couverture des tests :

```
php bin/phpunit --testdox --coverage-html tests/CoverageReport.
```

Documentation : <https://phpunit.de/documentation.html>.

J'intègre PHPStan dans le projet en exécutant la commande

```
composer require --dev phpstan/phpstan phpstan/extension-installer  
phpstan/phpstan-symfony phpstan/phpstan-doctrine.
```

Il s'installe avec un fichier de configuration (« phpstan.dist.neon ») par lequel on peut, entre autres, changer le niveau d'analyse - les extensions Symfony et Doctrine sont automatiquement configurées par l'extension-installer. Je fais un commit.

Pour lancer une analyse, on utilise `vendor/bin/phpstan analyze`.

Documentation : <https://phpstan.org/user-guide/getting-started>.

Pour PHP-CS-Fixer, bien que j'utilise une extension VS Code en local, je fais un

```
composer require --dev friendsofphp/php-cs-fixer
```

 pour le cas où d'autres personnes me rejoindraient sur le projet. Je le configure dans « .php-cs-fixer.dist.php » à la racine du projet. Je fais le commit correspondant. J'ai défini les règles suivantes :

```
.php-cs-fixer.dist.php > ...  
1  <?php  
2  
3  declare(strict_types=1);  
4  
5  $finder = (new PhpCsFixer\Finder())  
6      ->in(__DIR__)  
7      ->exclude('var')  
8      ->exclude('public/bundles')  
9      ->exclude('public/build')  
10     ->notPath('bin/console')  
11     ->notPath('public/index.php')  
12     ->notPath('importmap.php');  
13  
14  return (new PhpCsFixer\Config())  
15      ->setRules([  
16      '@Symfony' => true,  
17      '@Symfony:risky' => true,  
18      'declare_strict_types' => true,  
19      'linebreak_after_opening_tag' => true,  
20      'mb_str_functions' => true,  
21      'no_php4_constructor' => true,  
22      'no_unreachable_default_argument_value' => true,  
23      'no_useless_else' => true,  
24      'no_useless_return' => true,  
25      'php_unit_strict' => true,  
26      'phpdoc_order' => true,  
27      'strict_comparison' => true,  
28      'strict_param' => true,  
29      'blank_line_between_import_groups' => false,  
30  ])  
31      ->setFinder($finder)  
32      ->setCacheFile(__DIR__.'/var/.php-cs-fixer.cache');
```

Documentation : <https://github.com/PHP-CS-Fixer/PHP-CS-Fixer>.

La prochaine dépendance PHP que je souhaite installer et configurer est Webpack Encore : elle entre en conflit avec l'Asset Mapper de Symfony dans la gestion de certaines dépendances (notamment Symfony UX Turbo, Symfony UX Chart.js), aussi je choisis de le désinstaller, ainsi que les dépendances qui y sont liées par défaut. Je les réinstalle aussitôt en même temps que Webpack Encore (pour que leurs intégrations soient automatiquement gérées).

```
PowerShell
PS C:\> cd .\Projets\ecoride.site\
PS C:\Projets\ecoride.site> composer remove symfony/asset-mapper symfony/stimulus-bundle symfony/ux-turbo --quiet
PS C:\Projets\ecoride.site> composer require symfony/webpack-encore-bundle symfony/stimulus-bundle symfony/ux-turbo --quiet
PS C:\Projets\ecoride.site> npm install --force
npm warn using --force Recommended protections disabled.
(node:14648) ExperimentalWarning: CommonJS module C:\Users\Lox\scoop\apps\nodejs\23.1.0\node_modules\npm\node_modules\debug\src\node.js is loading ES Module C:\Users\Lox\scoop\apps\nodejs\23.1.0\node_modules\npm\node_modules\supports-color\index.js using require(). Support for loading ES Module in require() is an experimental feature and might change at any time (Use 'node --trace-warnings ...' to show where the warning was created)

up to date, audited 399 packages in 1s

56 packages are looking for funding
  run 'npm fund' for details

found 0 vulnerabilities
PS C:\Projets\ecoride.site> npm run build

> build
> encore production --progress

Running webpack ...

99% done plugins FriendlyErrorsWebpackPlugin DONE Compiled successfully in 1743ms
19:57:41

6 files written to public\build
Entrypoint app 175 KiB = runtime.8ab7f0c8.js 1.37 KiB 837.26db41c2.js 168 KiB app.b75294ae.css 30 bytes app.5193c17e.js 5.04 KiB
webpack compiled successfully
PS C:\Projets\ecoride.site> |
```

Je fais un `npm install` qui installe les dépendances JavaScript (notamment Encore).

Je peux alors générer les fichiers CSS et JavaScript avec `npm run build`. Puisque j'ai choisi de builder en local (avant de déployer en production), je supprime la ligne « /public/build/» du fichier « .gitignore ». La configuration de base de Webpack Encore est fournie dans « webpack.config.json », elle inclut une configuration de base pour Babel (transforme le code moderne en code compatible avec les anciens navigateurs) : je n'y touche pas pour l'instant. Je fais un commit des changements effectués.

Documentation : <https://symfony.com/doc/6.4/frontend/encore/index.html>.

La dernière dépendance PHP à ajouter est Symfony UX Chart.js : je le fais en exécutant `composer require symfony/ux-chartjs`, puis `npm install`. Je commit le tout.

Documentation : <https://symfony.com/bundles/ux-chartjs/current/index.html>.

e. Installation de dépendances CSS et JavaScript

La première dépendance que je veux installer est Bootstrap 5. Pour pouvoir en faciliter l'intégration et le personnalisable, je vais avoir besoin du support de Sass :

je commence par importer la bibliothèque Sass et le plugin Webpack sass-loader :

`npm install sass sass-loader --save-dev`, ensuite je décommente la ligne « `.enableSassLoader()` » dans « `webpack.config.js` » : Sass est maintenant pris en charge, je commit les changements.

Documentation : <https://symfony.com/doc/6.4/frontend/encore/css-preprocessors.html>.

Pour fonctionner, Bootstrap a besoin de Popper.js, j'installe les deux en même temps :

`npm install @popperjs/core bootstrap --save-dev`. Le style de Bootstrap s'importe dans les fichiers .scss par « `@import "~bootstrap/scss/bootstrap";` », les variables Bootstrap doivent être modifiées avant cet import pour être prises en compte. Le JavaScript de Bootstrap s'importe dans les fichiers .js par « `require('bootstrap');` ». Bootstrap s'intègre maintenant aux builds, je fais un commit.

Documentation : <https://symfony.com/doc/6.4/frontend/encore/bootstrap.html>.

Pour nettoyer les fichiers .css (enlever les classes inutilisées) et les rendre compatibles avec les anciens navigateurs, je veux utiliser postCSS en y intégrant deux plugins :

autoprefixer et PurgeCSS. J'importe les trois en même temps dans le projet, ainsi qu'un loader :

`npm install postcss postcss-loader autoprefixer @fullhuman/postcss-purgecss --save-dev`. Ensuite je crée un fichier « `.postcss.config.js` » à la racine du projet et ajoute une ligne « `.enablePostCssLoader()` » dans « `webpack.config.js` ». Je fais un commit des changements.

```
JS postcss.config.js > ...
1  export const plugins = {
2      autoprefixer: {},
3      "@fullhuman/postcss-purgecss": {
4          content: [
5              "./templates/*.html.twig",
6              "./templates/**/*.html.twig",
7              "./assets/*.js",
8              "./assets/**/*.js",
9          ],
10     },
11 },
12 };
```

Documentation : <https://symfony.com/doc/6.4/frontend/encore/postcss.html>.

Pour déterminer les navigateurs cibles (ceux sur lesquels je souhaite une prise en charge fonctionnelle), je complète le fichier « package.json » comme suit :

```
"browserslist": [  
  "> 0.5%",  
  "last 2 versions",  
  "Firefox ESR",  
  "not dead"  
]
```

Pour s'assurer que les dernières informations sur les navigateurs soient utilisées pour mettre à jour la base de données utilisée par « browserlist », j'utilise la commande `npx update-browserslist-db@latest` (npx est un outil de npm qui permet d'exécuter des paquets Node.js sans avoir besoin de les installer). C'est cette liste qui détermine le fonctionnement de autoprefixer et de Babel. Je commit cette liste.

Documentation : <https://github.com/browserslist/browserslist>.

La dernière dépendance Javascript que je veux intégrer au projet est ESLint : bien que j'utilise une extension VS Code en local, je fais `npm install eslint --save-dev` pour le cas où d'autres personnes me rejoindraient sur le projet et pour pouvoir l'intégrer au build. Ensuite j'exécute la commande `npm init @eslint/config@latest` qui pose une série de questions qui servent à rédiger le fichier « .eslint.config.mjs » qui configure le comportement d'ESLint.

J'ajoute un script dans « package.json » : « "lint": "eslint assets/**/*.js" », cela permet de lancer la commande `npm run lint`.

Ensuite, j'ajoute un plugin avec `npm install eslint-webpack-plugin --save-dev` afin d'exécuter ESLint automatiquement lors du build avec Webpack.

Je commit les changements

Documentation : <https://eslint.org/docs/latest/>.

8. Bases de données

Les systèmes de gestion de bases de données (SGBD) choisies sont :

- MySQL pour gérer les informations sur les utilisateurs, les véhicules et les données associées (relationnel).
- MongoDB pour gérer les données sur les trajets et les avis (non relationnel).

Pour faciliter le développement en local, des outils de visualisation et d'administration de données seront utiles. Je retiens des options éprouvées :

phpMyAdmin et Mongo Express, tous deux accessibles via un navigateur.

Ces quatre services seront déployés via des conteneurs Docker.

Par défaut, Symfony est livré avec Doctrine ORM qui permet de gérer des bases de données relationnelles de manière efficace. Pour compléter la configuration du projet et faciliter la gestion des bases de données NoSQL, il est nécessaire d'inclure Doctrine MongoDB ODM (Object Document Manager) en utilisant la commande suivante :

```
composer require doctrine/mongodb-odm-bundle.
```

Documentation : <https://www.doctrine-project.org/projects/doctrine-mongodb-odm/en/2.9/index.html>

Pour construire les services en local, Docker Compose est utilisé afin de simplifier leur configuration et leur déploiement. Le fichier « compose.yaml » décrit les quatre services (MySQL, phpMyAdmin, MongoDB et Mongo Express) avec les paramètres nécessaires pour qu'ils interagissent correctement entre eux.

Documentation : <https://docs.docker.com/reference/compose-file/>.

Pour choisir les images Docker à utiliser, je visite <https://hub.docker.com/> pour y trouver les images officielle (il y a un filtre de recherche disponible pour cela). Les versions « FPM » sont plus rapide, les « alpine » sont plus légères - l'onglet « Tags » présente les vulnérabilités connues pour chaque version proposée (analyse faite par Docker Scout).

Une fois le fichier « compose.yaml » rédigé (cf. page suivante) et Docker Desktop ouvert (ce qui lance le Docker daemon), j'exécute la commande `docker-compose up -d` qui lance les services en arrière-plan : cela démarre tous les conteneurs nécessaires au fonctionnement des base de données de l'application. Pour vérifier le bon déploiement : `docker-compose ps` affichera l'état de chaque service, les ports exposés et leur statut (informations également disponibles dans Docker Desktop). phpMyAdmin est accessible à l'adresse localhost:8080, Mongo Express est accessible à l'adresse localhost:8081.

La configuration des outils est terminée, j'exécute `git flow feature finish initial-configuration`.

```

compose.yml > ...
docker-compose.yml - The Compose specification establishes a standard for the definit
name: ecoride-stack

services:
  mysql:
    image: mysql:8.0
    container_name: mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: rootpassword
      MYSQL_DATABASE: ecoride_db
      MYSQL_USER: ecoride_user
      MYSQL_PASSWORD: userpassword
    ports:
      - "3306:3306"
    volumes:
      - mysql_data:/var/lib/mysql





























  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    container_name: phpmyadmin
    restart: always
    environment:
      PMA_HOST: mysql
      PMA_USER: ecoride_user
      PMA_PASSWORD: userpassword
    ports:
      - "8080:80"
    depends_on:
      - mysql

  mongodb:
    image: mongo:latest
    container_name: mongodb
    restart: always
    environment:
      MONGO_INITDB_DATABASE: ecoride_db
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: rootpassword
    ports:
      - "27017:27017"
    volumes:
      - mongodb_data:/data/db

  mongo-express:
    image: mongo-express
    container_name: mongodb-express
    restart: always
    environment:
      ME_CONFIG_MONGODB_ADMINUSERNAME: root
      ME_CONFIG_MONGODB_ADMINPASSWORD: rootpassword
      ME_CONFIG_MONGODB_SERVER: mongodb
    ports:
      - "8081:8081"
    depends_on:
      - mongodb

volumes:
  mysql_data:
  mongodb_data:

```

| <input type="checkbox"/> | Name | Image | Status | Port(s) | CPU (%) | Last started | Actions |
|--------------------------|--|---------------------------------------|------------------|---|---------|--------------|---|
| <input type="checkbox"/> |  ecoride-stack | | Running (4/4) | | 0.84% | 1 minute ago |    |
| <input type="checkbox"/> |  mysql 91e3b4fc7fe2  | mysql:8.0 | Running | 3306:3306  | 0.26% | 1 minute ago |    |
| <input type="checkbox"/> |  mongodb 305e2674380f  | mongo:latest | Running | 27017:27017  | 0.58% | 1 minute ago |    |
| <input type="checkbox"/> |  phpmyadmin 6859af20d5da  | phpmyadmin/phpmyadmin | Running | 8080:80  | 0% | 1 minute ago |    |
| <input type="checkbox"/> |  mongodb-express 2c08cc9fea2e  | mongo-express | Running | 8081:8081  | 0% | 1 minute ago |    |

9. Outils annexes

- Pour visualiser l'application, j'utilise plusieurs navigateurs : Google Chrome, Mozilla Firefox, Microsoft Edge et Opéra. Chacun d'eux propose des outils de développement intégrés.
- Pour la sécurité, j'utilise Bitdefender comme antivirus et me sers du VPN qu'il propose.
- Pour la gestion de projet, j'utilise Trello.

Téléchargement : <https://trello.com/platforms>.

- Pour la prise de note, j'utilise Notion.

Téléchargement : <https://www.notion.so/desktop>.

- Pour réaliser des diagrammes, j'utilise draw.io.

Téléchargement : <https://www.drawio.com/>.

- Pour le maquettage, j'utilise Figma.

Téléchargement : <https://www.figma.com/fr-fr/downloads/>.

- Pour la manipulation d'images, j'utilise gimp.

Téléchargement : <https://www.gimp.org/downloads/>.

- Pour la création des fichiers .svg, j'utilise Inkscape.

Téléchargement : <https://inkscape.org/fr/release/inkscape-1.4/>.

- Pour la vérifier l'accessibilité, j'utilise NVDA.

Téléchargement : <https://www.nvda.fr/c2>.

Tous ces outils sont également disponibles sur le Microsoft Store.