**Extension du Projet Mohammed NABI :**

**Guinot** **Le** 28/06/2022.

Table des matières

[**Veille effectuée sur les vulnérabilités de sécurité :** 3](#_Toc107346565)

[**Introduction :** 3](#_Toc107346566)

[**Importer des fichiers Csv en base de données :** 6](#_Toc107346567)

[**Documentation en Anglais :** 6](#_Toc107346568)

[**Traduction :** 10](#_Toc107346569)

# **Veille effectuée sur les vulnérabilités de sécurité :**

La sécurité est devenue l'un des points essentiels du développement informatique.

La ferme du web est un site de veille technologiques destinées aux développeurs, designers

et ergonomes web.

Créée en 2008, La Ferme du Web est une plateforme dédiée au partage des connaissances,

astuces, outils et bonnes pratiques du web / mobile.

Les thématiques sont variées et s'adressent particulièrement aux développeurs, webdesigners,

chefs de projets, webmasters et tous curieux du web.

On peut retrouvez la veille techno et créez nos alertes emails sur les sujets qui nous intéressent.

Le net, ça se cultive !

Remarque :

Une vulnérabilité a été découverte dans Symfony. Elle permet à un attaquant de provoquer une injection de requêtes illégitimes par rebond (CSRF).

SOLUTION :

Se référer au bulletin de sécurité de l'éditeur pour l'obtention des correctifs.

# **Introduction :**

Ce chapitre présente les bases des commandes Symfony.

Il prend pour exemple, la création d’une commande d’import de données à partir d’un fichier Csv.

Lien de la documentation officielle en Anglais de Symfony :

https://symfony.com/doc/current/console.html

<?php

namespace App\Command;

use App\Entity\User;

use Doctrine\ORM\EntityManagerInterface;

use Symfony\Component\Console\Command\Command;

use Symfony\Component\Console\Input\InputInterface;

use Symfony\Component\Console\Output\OutputInterface;

use Symfony\Component\Console\Style\SymfonyStyle;

use Symfony\Component\Security\Core\Encoder\UserPasswordEncoderInterface;

class ImportCommand extends Command

{

protected static $defaultName = 'app:import-command';

private string $csv; // La route du fichier csv est importée grâce au fichier services.yaml

private UserPasswordEncoderInterface $encoder;

private EntityManagerInterface $manager;

/\*\*

\* ImportCommand constructor.

\* @param string $csv

\* @param UserPasswordEncoderInterface $encoder

\* @param EntityManagerInterface $manager

\*/

public function \_\_construct(string $csv, UserPasswordEncoderInterface $encoder, EntityManagerInterface $manager)

{

parent::\_\_construct(self::$defaultName);

$this->csv = $csv;

$this->encoder = $encoder;

$this->manager = $manager;

}

protected function configure()

{

$this->setDescription('Import de 3 Utilisateurs à partir d\'un fichier csv');

}

protected function execute(InputInterface $input, OutputInterface $output): int

{

$io = new SymfonyStyle($input, $output);

if (($handle = fopen($this->csv, "r")) !== false) {

while (($line = fgetcsv($handle, 1000, ",")) !== false) {

$user = (new User())

->setFirstname($line[0])

->setLastname($line[1])

->setEmail($line[2]);

$user->setPassword($this->encoder->encodePassword($user, "motDePasse"));

$this->manager->persist($user);

}

fclose($handle);

}

$this->manager->flush();

$io->success('Votre commande est un franc succès');

return Command::SUCCESS;

}

}

# **Importer des fichiers Csv en base de données :**

## **Documentation en Anglais :**

Lien de la documentation officielle en Anglais de Symfony :

https://symfony.com/doc/current/console.html

Console Commands

Console Commands

Edit this page

Running Commands

• APP\_ENV & APP\_DEBUG

• Console Completion

Creating a Command

• Configuring the Command

• Registering the Command

• Running the Command

Console Output

• Output Sections

Console Input

Getting Services from the Service Container

Command Lifecycle…

use Symfony\Component\Console\Input\InputArgument;

// ...

protected function configure(): void

{

$this

// configure an argument

->addArgument('username', InputArgument::REQUIRED, 'The username of the user.')

// ...

;

}

// ...

public function execute(InputInterface $input, OutputInterface $output): int

{

$output->writeln([

'User Creator',

'============',

'',

]);

// retrieve the argument value using getArgument()

$output->writeln('Username: '.$input->getArgument('username'));

return Command::SUCCESS;

}

Now, you can pass the username to the command:

$ php bin/console app:create-user Wouter

User Creator

============

Username: Wouter

See also

Read Console Input (Arguments & Options) for more information about console options and arguments.

#

Getting Services from the Service Container

To actually create a new user, the command has to access some services. Since your command is already registered as a service, you can use normal dependency injection. Imagine you have a App\Service\UserManager service that you want to access:

Copy

// ...

use App\Service\UserManager;

use Symfony\Component\Console\Command\Command;

class CreateUserCommand extends Command

{

private $userManager;

public function \_\_construct(UserManager $userManager)

{

$this->userManager = $userManager;

parent::\_\_construct();

}

// ...

protected function execute(InputInterface $input, OutputInterface $output): int

{

// ...

$this->userManager->create($input->getArgument('username'));

$output->writeln('User successfully generated!');

return Command::SUCCESS;

}

}

#

Command Lifecycle

Commands have three lifecycle methods that are invoked when running the command:

initialize() (optional)

This method is executed before the interact() and the execute() methods. Its main purpose is to initialize variables used in the rest of the command methods.

interact() (optional)

This method is executed after initialize() and before execute(). Its purpose is to check if some of the options/arguments are missing and interactively ask the user for those values. This is the last place where you can ask for missing options/arguments. After this command, missing options/arguments will result in an error.

execute() (required)

This method is executed after interact() and initialize(). It contains the logic you want the command to execute and it must return an integer which will be used as the command exit status.

#

Testing Commands

Symfony provides several tools to help you test your commands. The most useful one is the CommandTester class. It uses special input and output classes to ease testing without a real console:

// tests/Command/CreateUserCommandTest.php

namespace App\Tests\Command;

use Symfony\Bundle\FrameworkBundle\Console\Application;

use Symfony\Bundle\FrameworkBundle\Test\KernelTestCase;

use Symfony\Component\Console\Tester\CommandTester;

class CreateUserCommandTest extends KernelTestCase

{

public function testExecute()

{

$kernel = self::bootKernel();

$application = new Application($kernel);

$command = $application->find('app:create-user');

$commandTester = new CommandTester($command);

$commandTester->execute([

// pass arguments to the helper

'username' => 'Wouter',

// prefix the key with two dashes when passing options,

// e.g: '--some-option' => 'option\_value',

]);

$commandTester->assertCommandIsSuccessful();

// the output of the command in the console

$output = $commandTester->getDisplay();

$this->assertStringContainsString('Username: Wouter', $output);

// ...

}

}

If you are using a single-command application, call setAutoExit(false) on it to get the command result in CommandTester.

## **Traduction :**

Nous allons voir comment créer sa propre commande avec Symfony.

Une commande permet de communiquer avec nos entités et nos services

la manipulation exclusive de l’administrateur. Elle permet d’éviter la création d’une interface graphique dans le cas où ce n’est pas nécessaire.

Quelques exemples :

• Importer des fichiers Csv en base de données

• Créer une entrée en base, un utilisateur par exemple

• Script de mise à jour

On commence ?

php bin/console make:command

ImportCommand.php

Notre mission est d’importer nos utilisateurs qui sont dans un fichier Csv

Le fichier CSV est dans SRC/Data ;

La fonction execute, est le corps de notre commande.

Voici le code pour un import simple de données en base.

protected function execute(InputInterface $input, OutputInterface $output): int

{

$io = new SymfonyStyle($input, $output);

if (($handle = fopen($this->csv, "r")) !== false) {

while (($line = fgetcsv($handle, 1000, ",")) !== false) {

$user = (new User())

->setFirstname($line[0])

->setLastname($line[1])

->setEmail($line[2]);

$user->setPassword($this->encoder->encodePassword($user, "motDePasse"));

$this->manager->persist($user);

}

fclose($handle);

}

$this->manager->flush();

$io->success('Votre commande est un franc succès');

return Command::SUCCESS;

}

Ici nous avons utilisé deux services externes à la commande, l’encodeur de mot de passe et l’Entity Manager, il suffit de les déclarer dans notre classe et de les définir dans le constructeur.

On modifie la description dans la fonction configure, et votre première commande Symfony est créée et fonctionnelle.

Pour finir…

Vous n’avez plus qu’à vous laisser emporter par la multitude d’options que vous offre cet outil. La documentation officielle (https://symfony.com/doc/current/console.html) vous explique aussi étape par étape la création, d’une commande mais dans la langue de Shakespeare.