|  |  |  |
| --- | --- | --- |
| **AI2** | Bartman Aleksander  Album 46466  Grupa 3 | **Wersja 1** |
|  |
| **LAB A** |

Automatyzacja pracy programisty WWW – Composer

# Spis treści

[Spis treści 1](#_Toc146276108)

[Cel zajęć 1](#_Toc146276109)

[Rozpoczęcie 1](#_Toc146276110)

[Jak wypełnić to sprawozdanie? 1](#_Toc146276111)

[Pobranie i uruchomienie PHP 2](#_Toc146276112)

[Inicjalizacja projektu z wykorzystaniem Composer 3](#_Toc146276113)

[Utworzenie i wykonanie programu 5](#_Toc146276114)

[Zarządzanie zależnościami 6](#_Toc146276115)

[Wykorzystanie różnych poziomów logowania Monolog 8](#_Toc146276116)

[Dependency Injection 9](#_Toc146276117)

[Podsumowanie 10](#_Toc146276118)

# Cel zajęć

Celem głównym zajęć jest zdobycie umiejętności tworzenia projektów oraz łączenia zależności z wykorzystaniem narzędzia Composer.

# Rozpoczęcie

Rozpoczęcie zajęć. Przedstawienie prowadzącego. Przedstawienie uczestników. Przedstawienie zasad laboratorium.

# Jak wypełnić to sprawozdanie?

Zapisz ten plik na dysku twardym jako kopię. Zmień nazwę pliku:

* grN na odpowiedni numer grupy (np. gr3),
* nazwisko-imie na Twoje dane bez polskich znaków.

Otwórz kolejno Plik -> Informacje -> Właściwości -> Właściwości zaawansowane -> Niestandardowe.

Zaktualizuj właściwości:



Czytaj tę instrukcję, wypełniaj polecenia, uzupełniaj zrzuty ekranu zgodnie z poleceniami.

Gotowe sprawozdanie wyślij w nieprzekraczalnym terminie **w postaci pliku PDF**.

# Pobranie i uruchomienie PHP

Zaloguj się do systemu Windows / pulpitu zdalnego rdp.wi.zut.edu.pl:

* spoza sieci ZUT potrzebny VPN: <https://uci.zut.edu.pl/uslugi-uci/vpn.html>;
* nazwa użytkownika: WIAD\ab12345
* komputer: rdp.wi.zut.edu.pl

Utwórz katalog I:\AI2-lab. Jeśli musisz umieścić ten folder gdzie indziej – upewnij się, że nie ma spacji i ogonków.

Odwiedź stronę <https://windows.php.net/download/>. Pobierz PHP 8.2.10 x64 NTS.

Wypakuj pobrane repozytorium do I:\AI2-lab\php-8.2.10-nts-Win32-vs16-x64.

Otwórz panel sterowania. W polu wyszukiwania wpisz path. Wybierz edycję zmiennych środowiskowych użytkownika. Znajdź zmienną Path i kliknij edycję. Dodaj ścieżkę I:\AI2-lab\php-8.2.10-nts-Win32-vs16-x64.

Skopiuj plik I:\AI2-lab\php-8.2.10-nts-Win32-vs16-x64\php.ini-development jako php.ini, po czym edytuj jego zawartość – odkomentuj poniższe ustawienia:

extension\_dir = "ext"

...

extension=curl

extension=gd

extension=intl

extension=mbstring

extension=openssl

extension=pdo\_sqlite

Utwórz katalog I:\AI2-lab\labA.

Otwórz ulubiony terminal (CMD, wiersz polecenia, PowerShell, Git Bash) i wejdź do katalogu I:\AI2-lab\labA.

Wykonaj komendę

php -i | Select-String -Pattern '(PHP Version)|(extension\_dir)|(OpenSSL support)|(PDO drivers)|(GD Support)|intl|(cURL support)|multibyte'

Oczekiwany wynik:



Zastąp poniższy obrazek swoim zrzutem ekranu:

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Opis wygenerowany automatycznie

|  |  |  |
| --- | --- | --- |
| Punkty: | 0 | 1 |

# Inicjalizacja projektu z wykorzystaniem Composer

Przejdź terminalem i eksploratorem plików do katalogu I:\AI2-lab\labA.

Pobierz archiwum PHAR composera w wersji 2.6.3 do katalogu I:\AI2-lab\labA:

* <https://getcomposer.org/download/2.6.3/composer.phar>

Sprawdź wersję composera:

php composer.phar --version

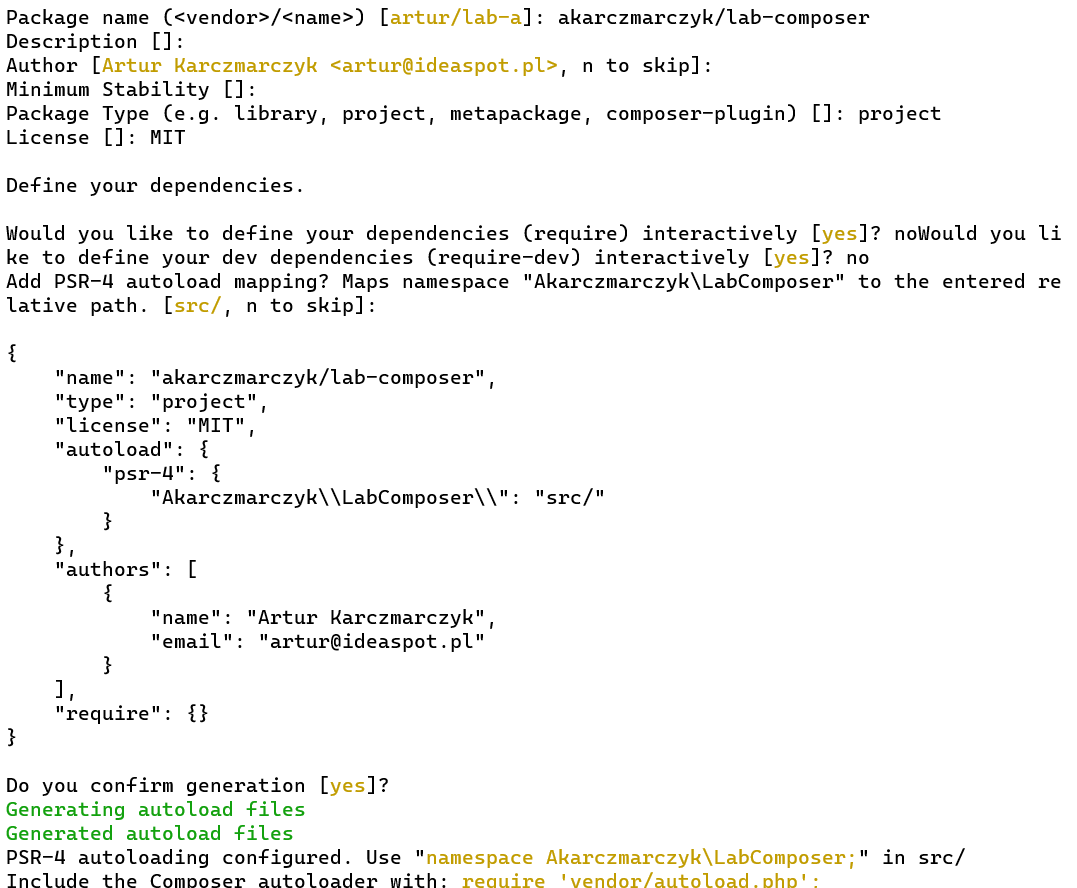
Zainicjuj projekt:

php composer.phar init

Ważne ustawienia:

* package name: inazwisko/lab-composer
* author: Imie Nazwisko
* package type: project
* license: MIT
* interaktywne wyszukiwanie pakietów: nie
* PSR-4 mapping: ENTER (zostawić domyślne)

Zweryfikuj ustawienia i zatwierdź utworzenie projektu:



Na koniec wykonaj polecenie:

php composer.phar install

Umieść poniżej zrzut ekranu z procesu inicjalizacji projektu composerem:

Obraz zawierający tekst, oprogramowanie, Czcionka, Strona internetowa

Opis wygenerowany automatycznie

Umieść poniżej zrzut ekranu przedstawiający otrzymaną strukturę katalogów, z wykorzystaniem polecenia:

Get-ChildItem -Path . -Recurse -Force -ErrorAction SilentlyContinue | Select-Object FullName

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

|  |  |  |
| --- | --- | --- |
| Punkty: | 0 | 1 |

# Utworzenie i wykonanie programu

Otwórz katalog I:\AI2-lab\labA za pomocą Visual Studio Code lub PhpStorm. Utwórz dwa pliki, zmieniając odpowiednio przestrzenie nazw:

<?php // src/Duck.php

namespace Akarczmarczyk\LabComposer;

class Duck

{

public function quack()

{

echo "Quack\n";

}

}

<?php // index.php

require\_once \_\_DIR\_\_ . '/vendor/autoload.php';

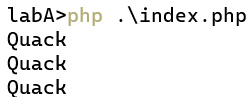
$duck = new \Akarczmarczyk\LabComposer\Duck();

$duck->quack();

$duck->quack();

$duck->quack();

Następnie z poziomu terminala wykonaj program:



Umieść poniżej zrzut ekranu pliku index.php:

Obraz zawierający tekst, Czcionka, oprogramowanie, Oprogramowanie multimedialne

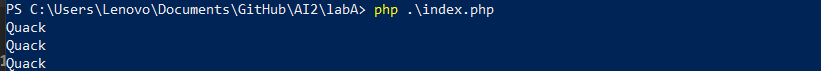
Opis wygenerowany automatycznie

Umieść poniżej zrzut ekranu pliku src/Duck.php:

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Umieść poniżej zrzut ekranu wywołania i działania programu (komenda php index.php):



|  |  |  |
| --- | --- | --- |
| Punkty: | 0 | 1 |

# Zarządzanie zależnościami

Przyjrzyj się strukturze programu, w szczególności plikom composer.json i composer.lock. Oba pliki muszą być commitowane. Dlaczego?

Zapoznaj się z zawartością katalogu vendor. Katalog vendor nie powinien być commitowany. Dlaczego?

Zainstaluj i odinstaluj pakiet monolog/monolog. Zbadaj jak zmienia się zawartość composer.json, composer.lock i katalogu vendor.

php composer.phar require monolog/monolog

php composer.phar remove monolog/monolog

Ponownie zainstaluj pakiet monolog/monolog:

php composer.phar require monolog/monolog

Po czym skasuj katalog vendor i spróbuj uruchomić program. Czy działa?

php index.php

Wykonaj composer install i zbadaj zawartość katalogu vendor. Spróbuj ponownie uruchomić program.

Omów jak zmienia się zawartość plików composer.json, composer.lock i katalogu vendor:

a) po zainstalowaniu pakietu monolog/monolog:

W pliku composer.json została dodana nowa zależność Monologa oraz wersje, którą Composer może zainstalować.

W pliku composer.lock znajdą się szczegółowe informacje o wersjach Monologa i jego zależności, które zostały zainstalowane.

Katalog vendor został dodany folder monolog, w którym znajduje się np. Kod źródłowy Monologa, katalog src (pliki źródłowe Monologa), pliki konfiguracyjne Monologa, pliki README oraz dokumentacja oraz pliki zależności Monologa i inne

b) po odinstalowaniu pakietu monolog/monolog:

W pliku composer.json została usunięta zależność Monologa oraz jego wersja.

W pliku composer.lock zostały usunięte informacje o Monologu oraz jego zależnościach.

W katalogu vendor został usunięty katalog Monolog wraz z całą zawartością.

Dlaczego po zainstalowaniu pakietu monolog/monolog i skasowaniu katalogu vendor aplikacja przestała się uruchamiać? Wyjaśnij. Umieść zrzut ekranu.

Dzieje się tak, ponieważ katalog „vendor” zawiera wszystkie zależności i biblioteki, które są niezbędne do prawidłowego działania aplikacji. Po usunięciu katalogu aplikacja nie może znaleźć wymaganych plików i klas co prowadzi do błędów.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

|  |  |  |
| --- | --- | --- |
| Punkty: | 0 | 1 |

# Wykorzystanie różnych poziomów logowania Monolog

W tej części wykorzystamy bibliotekę monolog do logowania komunikatów i błędów. Zmodyfikuj kod index.php, żeby dodać logowanie do pliku monolog.log:

<?php // index.php

require\_once \_\_DIR\_\_ . '/vendor/autoload.php';

$ds = DIRECTORY\_SEPARATOR;

$log = new \Monolog\Logger("my\_log");

$log->pushHandler(new \Monolog\Handler\StreamHandler(\_\_DIR\_\_ . $ds . 'monolog.log', \Monolog\Logger::ERROR));

$log->error("This is some error.");

$log->warning("This is some warning.");

$log->notice("This is some notice.");

$log->info("This is some info.");

$log->debug("This is some debug.");

$duck = new \Akarczmarczyk\LabComposer\Duck();

$duck->quack();

$duck->quack();

$duck->quack();

Uruchom program:

php index.php

Sprawdź zawartość pliku monolog.log.

Kolejno zmieniaj ERROR w kodzie na WARNING, NOTICE, INFO i DEBUG i uruchamiaj program. Omów wpływ zmiany na liczbę zapisywanych logów. Omów korzyści praktyczne płynące z umieszczania funkcji logujących w całym programie i przełączania poziomu logów w jednym miejscu.

Po każdej zmianie poziomu logowania pojawiały się nowe rodzaje wpisów na początku były same ERROR-y później zostały dodane jeszcze wpisy WARNING-ów, przy NOTICE zostały dodane wpisy dotyczące notatek itd.

Korzyści jakie płyną z takiego rozwiązania to przede wszystkim:

-łatwa kontrola nad tym jakie informacje są logowane bez konieczności modyfikowania kodu w wielu miejscach

-można łatwo przełączać poziomy logów co przyśpiesza prace podczas np. debugowania można przełączyć na DEBUG, aby uzyskać szczegółowe informacje o działaniu programu.

-gromadzenie logów w jednym miejscu przyśpiesza czas analizowania oraz debugowania aplikacji, ponieważ wszystko co jest nam potrzebne jest w jednym pliku

Wstaw reprezentatywny fragment pliku monolog.log:

Obraz zawierający tekst, zrzut ekranu, Czcionka, oprogramowanie

Opis wygenerowany automatycznie

|  |  |  |
| --- | --- | --- |
| Punkty: | 0 | 1 |

# Dependency Injection

Dependency injection to technika programistyczna wchodząca w skład architektury heksagonalnej, który umożliwia uniezależnienie klasy od jej zależności. W tej sekcji wstrzykniemy do klasy Duck loggera, aby można było logować zdarzenia w instancjach tej klasy.

Zmodyfikuj klasę Duck.php, żeby:

* wykorzystywała technikę Dependency Injection do przekazywania obiektu klasy LoggerInterface;
* logowała (poziom INFO) stworzenie klasy Duck;
* logowała (poziom DEBUG) wykonanie metody Duck::quack().

Następnie przetestuj uruchamianie kodu źródłowego z poziomami logowania ERROR, WARNING, INFO, DEBUG.

<?php // src/Duck.php

namespace Akarczmarczyk\LabComposer;

use Psr\Log\LoggerInterface;

class Duck

{

/\*\* @var LoggerInterface \*/

private $logger;

public function \_\_construct(LoggerInterface $logger = null)

{

$this->logger = $logger;

if ($this->logger) {

$this->logger->info("Duck created.");

}

}

public function quack()

{

if ($this->logger) {

$this->logger->debug("Quack() executed.");

}

echo "Quack\n";

}

}

<?php // index.php

require\_once \_\_DIR\_\_ . '/vendor/autoload.php';

$ds = DIRECTORY\_SEPARATOR;

$log = new \Monolog\Logger("my\_log");

$log->pushHandler(new \Monolog\Handler\StreamHandler(\_\_DIR\_\_ . $ds . 'log.log', \Monolog\Logger::ERROR));

$log->error("error");

$log->warning("warning");

$duck = new \Akarczmarczyk\LabComposer\Duck($log);

$duck->quack();

$duck->quack();

$duck->quack();

Kolejno zmieniaj ERROR w kodzie na WARNING, NOTICE, INFO i DEBUG i uruchamiaj program. Omów wpływ zmiany na liczbę zapisywanych logów. Zamieść odpowiedni wycinek pliku monolog.log:

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Dzięki dodaniu konstruktora z argumentem obiektu klasy „LoggerInterface” możemy przekazywać logera do istancji klasy „Duck” z zewnątrz (mechanizm Dependency Injection). Klasa Duck teraz loguje informacje o swoim utworzeniu (poziom INFO) oraz wykonaniu metody quack() (poziom DEBUG). Poziom logowania można kontrolować z poziomu pliku index.php.

|  |  |  |
| --- | --- | --- |
| Punkty: | 0 | 1 |

# Podsumowanie

W kilku zdaniach podsumuj zdobyte podczas tego laboratorium umiejętności.

Podczas tego laboratorium nauczyłem się inicjalizować projekt za pomocą narzędzia Composer. Zaznajomiłem się z budową projektu oraz zawartością ważnych plików takich jak np. composer.json i composer.lock oraz katalogu vendor. Zarządzaniem zależnościami w projekcie za pomocą Composer (dodawanie oraz usuwanie pakietów). Wykorzystania różnych poziomów logowania przy użyciu biblioteki Monolog. Jak zaimplementować mechanizm Dependency Injection do klasy „Duck” co pozwoliło mi na przekazywanie logera jako zależności do klasy.

Zweryfikuj kompletność sprawozdania. Utwórz PDF i wyślij w terminie.