|  |  |  |
| --- | --- | --- |
| **AI2** | Bancewicz Aleksandra  Album ba34753  Grupa 1 | **Wersja 1** |
|  |
| **LAB A** |

Automatyzacja pracy programisty WWW – Composer

# Spis treści

[Spis treści 1](#_Toc178111744)

[Cel zajęć 1](#_Toc178111745)

[Rozpoczęcie 1](#_Toc178111746)

[Jak wypełnić to sprawozdanie? 1](#_Toc178111747)

[Pobranie i uruchomienie PHP 2](#_Toc178111748)

[Inicjalizacja projektu z wykorzystaniem Composer 3](#_Toc178111749)

[Utworzenie i wykonanie programu 5](#_Toc178111750)

[Zarządzanie zależnościami 7](#_Toc178111751)

[Wykorzystanie różnych poziomów logowania Monolog 9](#_Toc178111752)

[Dependency Injection 11](#_Toc178111753)

[Podsumowanie 13](#_Toc178111754)

# 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:

1

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.

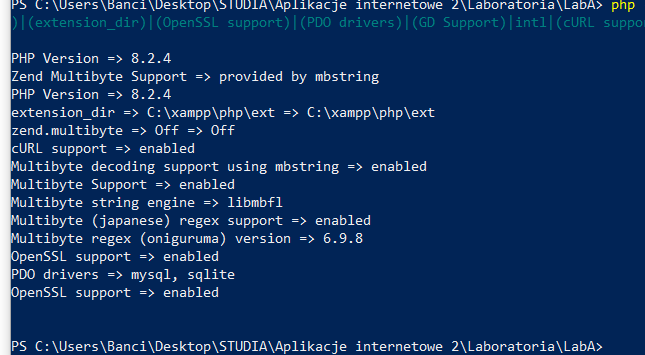
Wykonaj komendę

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:



|  |  |  |
| --- | --- | --- |
| 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

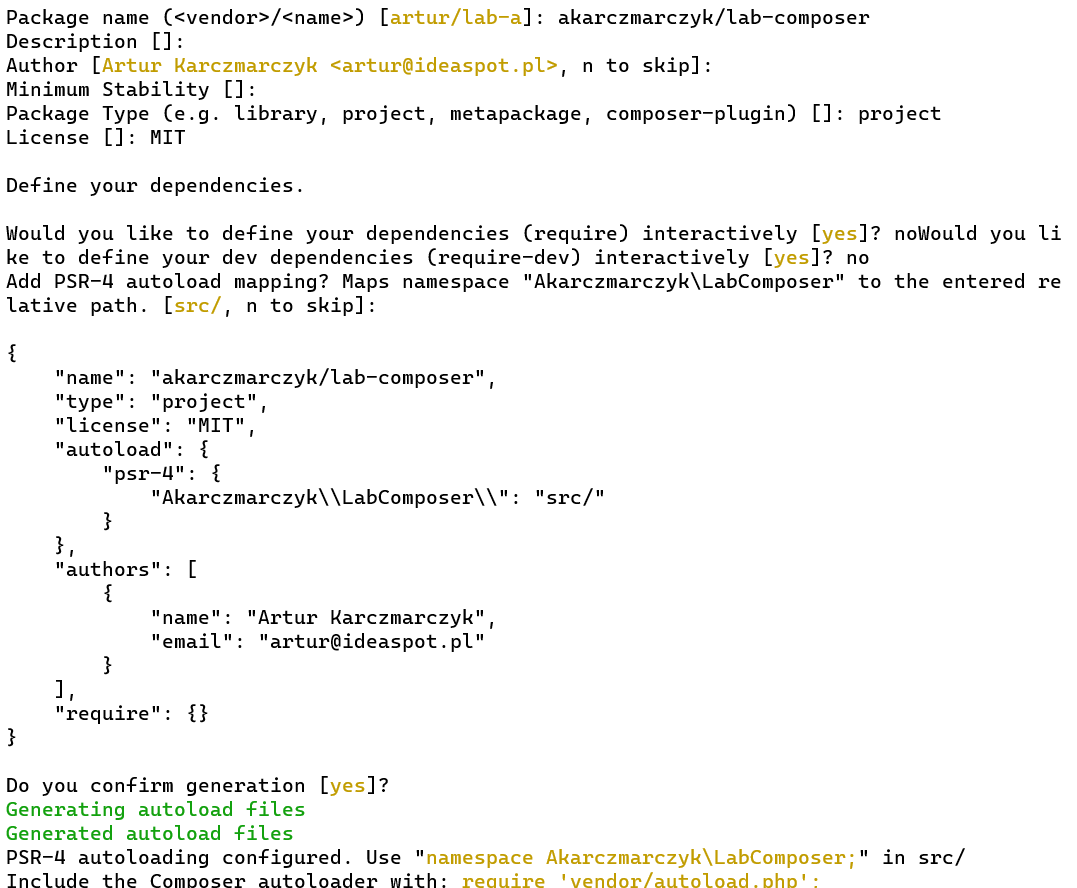
Zainicjuj projekt:

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)

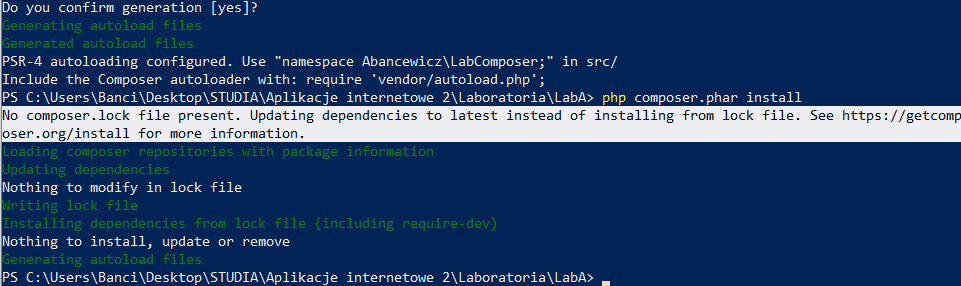
Zweryfikuj ustawienia i zatwierdź utworzenie projektu:



Na koniec wykonaj polecenie:

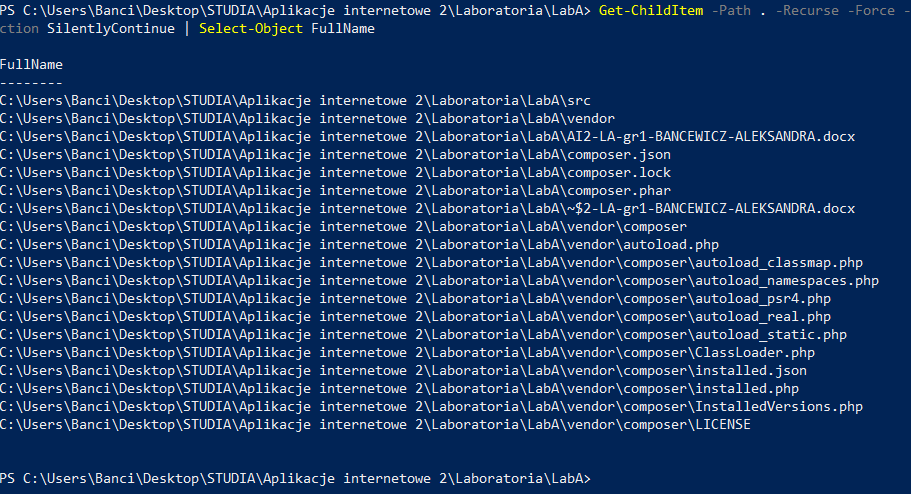
php composer.phar install

Umieść poniżej zrzut ekranu z procesu inicjalizacji projektu composerem:



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

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



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

# Utworzenie i wykonanie programu

Otwórz katalog I:\AI2-lab\labA za pomocą 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';

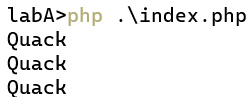
$duck = new \Akarczmarczyk\LabComposer\Duck();

$duck->quack();

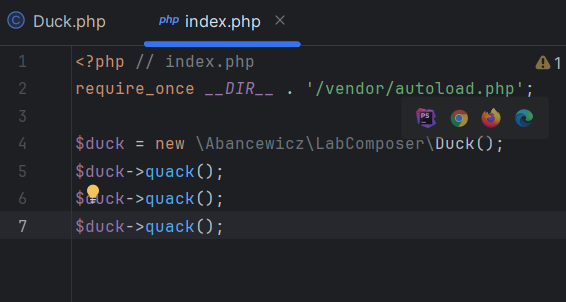
$duck->quack();

$duck->quack();

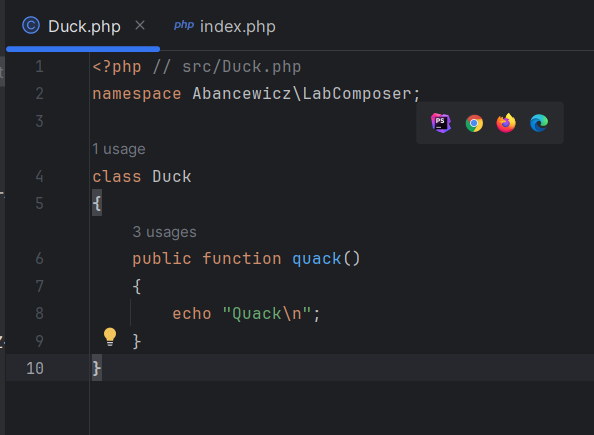
Następnie z poziomu terminala wykonaj program:



Umieść poniżej zrzut ekranu pliku index.php:



Umieść poniżej zrzut ekranu pliku src/Duck.php:



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

Obraz zawierający tekst, oprogramowanie, Oprogramowanie multimedialne, Czcionka

Opis wygenerowany automatycznie

|  |  |  |
| --- | --- | --- |
| 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:

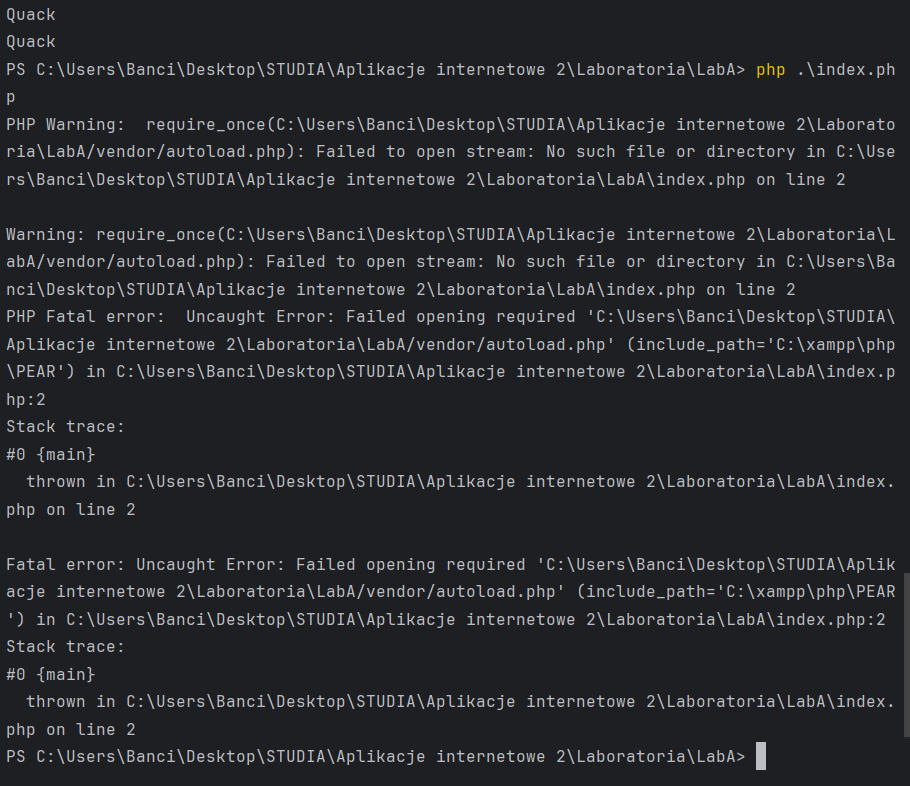
Composer dodaje nowy wpis w sekcji "require" pliku composer.json

b) po odinstalowaniu pakietu monolog/monolog:

wpis dotyczący monolog/monolog zostaje usunięty z sekcji "require"

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

Katalog vendor zawiera wszystkie pliki zależności potrzebne do działania aplikacji, w tym biblioteki zewnętrzne, autoloader Composera (autoload.php), a także wszystkie zainstalowane pakiety, w tym monolog/monolog. Gdy katalog vendor zostanie usunięty, aplikacja traci dostęp do tych zależności, w tym do autoloadera.  
W efekcie, aplikacja nie będzie mogła się uruchomić, ponieważ brakuje jej podstawowych komponentów, które są przechowywane w katalogu vendor.



|  |  |  |
| --- | --- | --- |
| 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.

Monolog, który jest używany do logowania w aplikacjach PHP, obsługuje różne poziomy logów. Zmienianie poziomu logowania wpływa na to, jakie informacje są zapisywane w logach.

**ERROR**: Zapisuje błędy, które mogłyby spowodować awarię aplikacji lub jej nieprawidłowe działanie. To poważne sytuacje, które wymagają natychmiastowej uwagi.

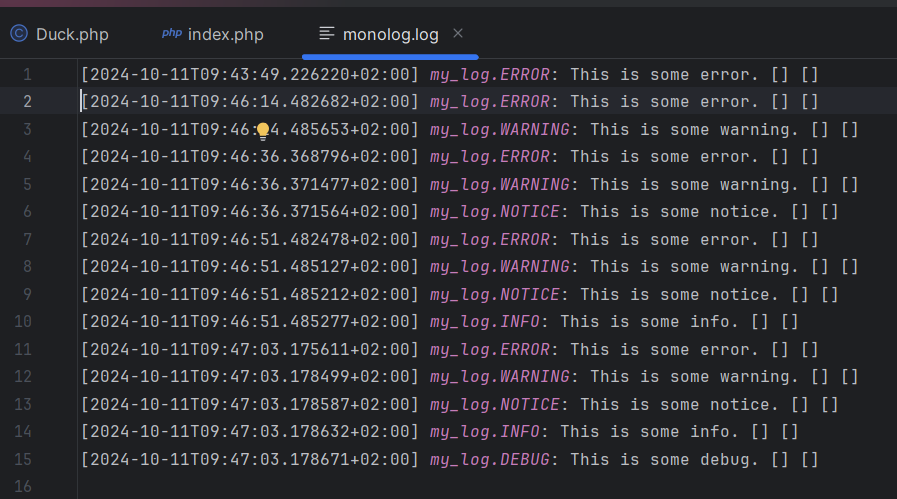
**WARNING**: Zapisuje ostrzeżenia o problemach, które mogą prowadzić do błędów, ale na razie nie powodują krytycznych skutków. Warto je monitorować.

**NOTICE**: Mniej istotne niż ostrzeżenia, ale nadal ważne, np. informacje o mniej poważnych błędach lub nieefektywnym kodzie.

**INFO**: Zapisuje ogólne informacje o działaniu aplikacji, które mogą być przydatne do analizy, ale nie są krytyczne.

**DEBUG**: Zapisuje szczegółowe informacje, które są pomocne podczas debugowania aplikacji, np. dane wejściowe, zmienne, kontrola przepływu kodu.

Wstaw reprezentatywny fragment pliku monolog.log:



|  |  |  |
| --- | --- | --- |
| 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:



Zmieniając poziom logowania z ERROR na WARNING, NOTICE, INFO i DEBUG wpływam na to, które logi są zapisywane w pliku. Monolog rejestruje logi tylko dla poziomu, który jest równy lub bardziej szczegółowy niż ustawiony w handlerze.

Monolog ma 5 poziomów logowania (od najbardziej szczegółowego do najbardziej ogólnego):

DEBUG — najniższy poziom, loguje szczegóły, które są przydatne w czasie debugowania.

INFO — używane do informacyjnych komunikatów, które mogą być pomocne w analizie aplikacji, ale nie są błędami.

NOTICE — mniej istotne komunikaty niż INFO, ale mogą wskazywać na potencjalny problem.

WARNING — ostrzeżenia o potencjalnych problemach, które nie są błędami.

ERROR — błędy, które mogą wpływać na działanie aplikacji.

CRITICAL, ALERT, EMERGENCY — poziomy wyższe niż ERROR (nie są używane w tym przykładzie).

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

# Podsumowanie

W kilku zdaniach podsumuj zdobyte podczas tego laboratorium umiejętności.

Podczas tego laboratorium zdobyłam umiejętności związane z zarządzaniem zależnościami w PHP przy użyciu narzędzia Composer, w tym instalowaniem i odinstalowywaniem pakietów, a także rozumieniem roli plików composer.json i composer.lock. Nauczyłam się również, jak korzystać z technik logowania, w tym integracji Monolog z aplikacją, a także jak zastosować technikę Dependency Injection do wstrzykiwania zależności, takich jak logger, do klas. Poznałam różnice między poziomami logowania (np. ERROR, WARNING, INFO, DEBUG) i ich wpływ na ilość i rodzaj zapisywanych logów. Dzięki tym doświadczeniom zyskałam większą biegłość w organizowaniu i kontrolowaniu zależności oraz logowania w aplikacjach PHP.

Zweryfikuj kompletność sprawozdania. Utwórz PDF i wyślij w terminie.