

SERWISY I KOMENDY

SPIS TREŚCI

Spis treści	1
Cel zajęć.....	1
Rozpoczęcie	1
Uwaga	1
Serwis WeatherUtil	1
Komendy	4
Commit projektu do GIT.....	7
Podsumowanie.....	7

CEL ZAJĘĆ

Celem głównym zajęć jest zdobycie następujących umiejętności:

- zamykanie reużywalnej logiki biznesowej w serwisach;
- wykorzystanie serwisów w kontrolerach;
- wykorzystanie serwisów w komendach;
- tworzenie komend konsolowych.

ROZPOCZĘCIE

Rozpoczęcie zajęć. Powtórzenie zasad tworzenia serwisów, komend.

Wejściówka?

UWAGA

Ten dokument aktywnie wykorzystuje niestandardowe właściwości. Podobnie jak w LAB A wejdź do Plik -> Informacje -> Właściwości -> Właściwości zaawansowane -> Niestandardowe i zaktualizuj pola. Następnie uruchom ten dokument ponownie lub Ctrl+A -> F9.

SERWIS WEATHERUTIL

Utwórz nową klasę `src/Service/WeatherUtil.php`, a w niej deklaracje dwóch metod:

- `getWeatherForLocation($location)` – odpowiedzialna za pobranie pomiarów (prognoza pogody) na podstawie encji lokalizacji;
- `getWeatherForCountryAndCity($countryCode, $cityName)` – odpowiedzialna za pobranie pomiarów na podstawie kodu kraju i nazwy miasta. Wewnętrzna implementacja sprowadza się do pobrania lokalizacji na podstawie kodu kraju i nazwy miasta, a następnie wywołania metody `getWeatherForLocation` dla otrzymanej lokalizacji.

```
<?php
declare(strict_types=1);

namespace App\Service;

use App\Entity\Location;
use App\Entity\Measurement;

class WeatherUtil
{
    /**
     * @return Measurement[]
     */
    public function getWeatherForLocation(Location $location): array
    {
        return [];
    }

    /**
     * @return Measurement[]
     */
    public function getWeatherForCountryAndCity(string $countryCode, string $city): array
    {
        return [];
    }
}
```

Zmodyfikuj WeatherController, żeby wykorzystywał nowy serwis:

<pre>class WeatherController extends AbstractCon { #[Route('/weather/{country}/{city}', na public function city(#[MapEntity(mapping: ['country' => Location \$location, MeasurementRepository \$repository,): Response { \$measurements = \$repository->findBy return \$this->render('weather/city. 'location' => \$location, 'measurements' => \$measurements }); } }</pre>	<pre>12 13 13 14 14 15 15 16 16 17 17 18 >> 18 19 19 20 20 21 21 22 22 23 23 24 24 25 25 26 26 27 27 28 28 29</pre>	<pre>class WeatherController extends AbstractController { #[Route('/weather/{country}/{city}', name: 'app_weather', requ public function city(#[MapEntity(mapping: ['country' => 'country', 'city' => 'c Location \$location, WeatherUtil \$util,): Response { \$measurements = \$util->getWeatherForLocation(\$location); return \$this->render('weather/city.html.twig', ['location' => \$location, 'measurements' => \$measurements,]); } }</pre>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Sprawdź, czy strona prognozy pogody wciąż działa. Nie powinno być żadnych błędów, jednakże zwracana lista pomiarów będzie pusta.

Zaimplementuj ciało metody `getWeatherForLocation()`. Wstaw zrzut ekranu kodu:

```

1 usage
14 public function getWeatherForLocation(Location $location): array
15 {
16     return $this->measurementRepository->findBy([
17         'location' => $location,
18     ]);
19 }
20

```

Punkty:

0

1

Zaimplementuj ciało metody `getWeatherForCountryAndCity()`. Wstaw zrzut ekranu kodu:

```

no usages
37 public function getWeatherForCountryAndCity(string $countryCode, string $city): array
38 {
39     $location = $this->locationRepository->findOne([
40         'country' => $countryCode,
41         'city' => $city,
42     ]);
43
44     if (!$location) {
45         return [];
46     }
47
48     return $this->getWeatherForLocation($location);
49 }
50

```

Punkty:

0

1

Wstaw zrzut ekranu kodu kontrolera wykorzystującego metodę z serwisu:

```

final class WeatherController extends AbstractController
{
    #[Route('/weather/{city}', name: 'app_weather')]
    public function city(
        string $city,
        LocationRepository $locationRepository,
        WeatherUtil $util,
    ): Response
    {
        // Znajdź lokalizację na podstawie miasta
        $location = $locationRepository->findOneBy(['city' => $city]);

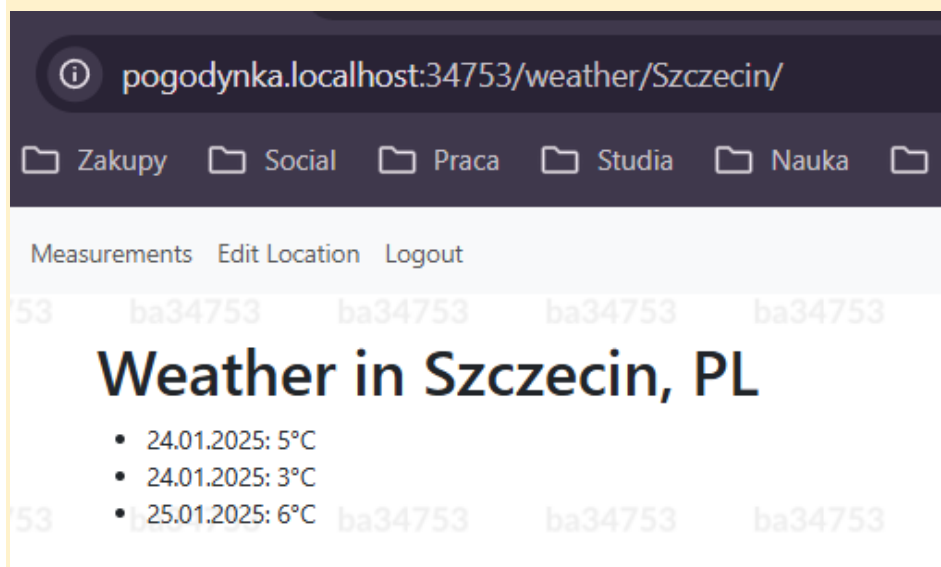
        if (!$location) {
            throw $this->createNotFoundException("City '$city' not found in the database.");
        }

        $measurements = $util->getWeatherForLocation($location);

        return $this->render( view: 'weather/city.html.twig', [
            'location' => $location,
            'measurements' => $measurements,
        ]);
    }
}

```

Wstaw zrzut ekranu prognozy pogody z pomiarami pobranymi z serwisu:



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

KOMENDY

Wykorzystaj komendę `make : command` do utworzenia komendy `weather:location`, służącej do pobierania prognozy pogody dla lokalizacji:

```
php .\bin\console make:command

Choose a command name (e.g. app:agreeable-pizza):
> weather:location

created: src/Command/WeatherLocationCommand.php

Success!

Next: open your new command class and customize it!
Find the documentation at https://symfony.com/doc/current/console.html
```

Edytuj utworzony plik `src/Command/WeatherLocationCommand.php`:

- podłącz serwis do konstruktora;
- ustaw wymagany argument id lokalizacji;
- w `execute()` wykorzystaj serwis do pobrania prognozy pogody;
- wydrukuj prognozę pogody na widok.

Przykładowe wywołanie:

```
php .\bin\console weather:location 1
Location: Szczecin
2023-09-26: 18
```

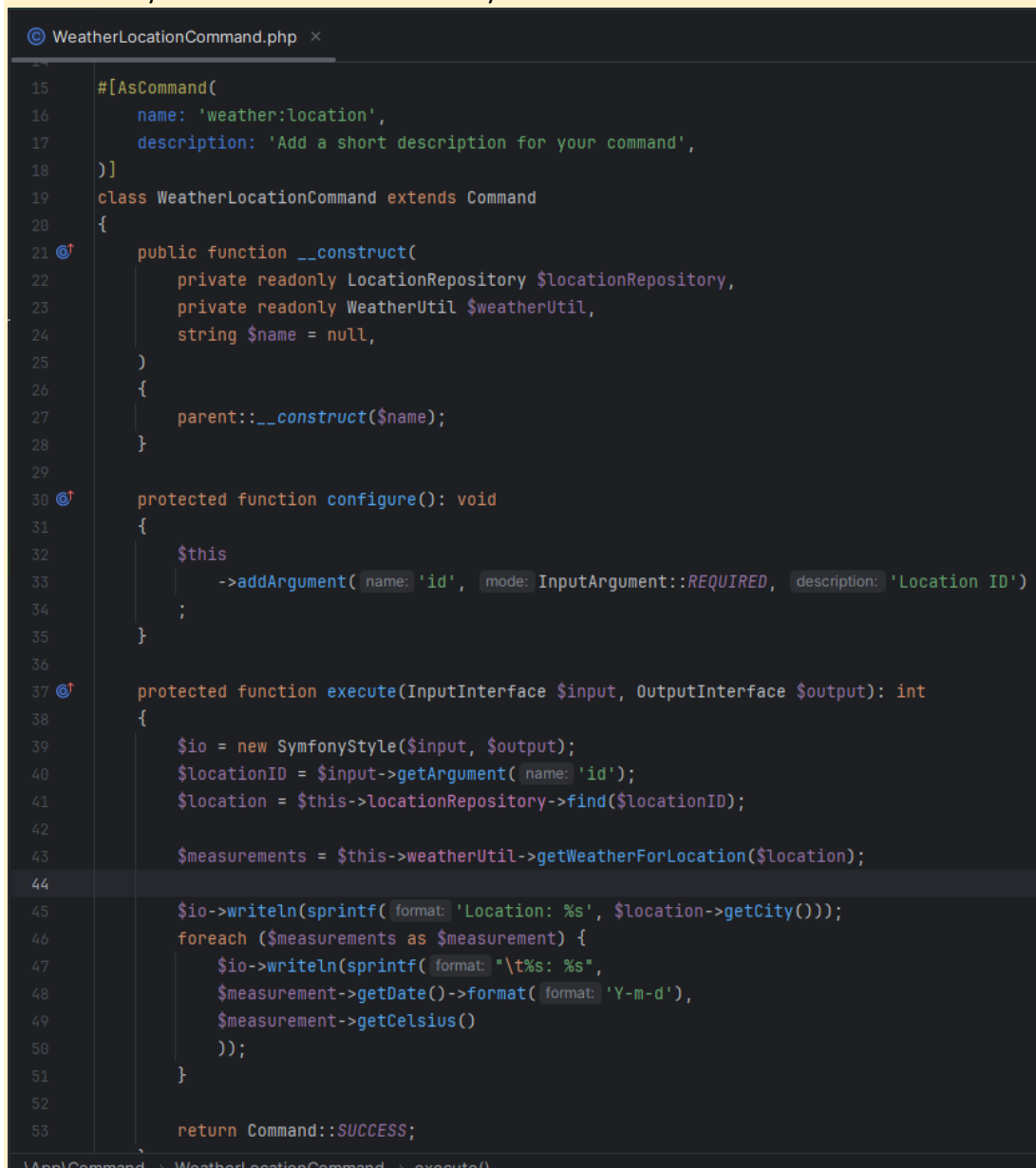
Przykładowy kod:

```
protected function execute(InputInterface $input, OutputInterface $output): int
{
    $io = new SymfonyStyle($input, $output);
    $locationId = $input->getArgument('id');
    $location = $this->locationRepository->find($locationId);

    $measurements = $this->weatherUtil->getWeatherForLocation($location);
    $io->writeln(sprintf('Location: %s', $location->getCity()));
    foreach ($measurements as $measurement) {
        $io->writeln(sprintf("\t%s: %s",
            $measurement->getDate()->format('Y-m-d'),
            $measurement->getCelsius()
        ));
    }

    return Command::SUCCESS;
}
```

Wstaw rzuty ekranu kodu całości komendy:



```

15  #[AsCommand(
16      name: 'weather:location',
17      description: 'Add a short description for your command',
18  )]
19  class WeatherLocationCommand extends Command
20  {
21      public function __construct(
22          private readonly LocationRepository $locationRepository,
23          private readonly WeatherUtil $weatherUtil,
24          string $name = null,
25      )
26      {
27          parent::__construct($name);
28      }
29
30      protected function configure(): void
31      {
32          $this
33              ->addArgument( name: 'id', mode: InputArgument::REQUIRED, description: 'Location ID')
34              ;
35      }
36
37      protected function execute(InputInterface $input, OutputInterface $output): int
38      {
39          $io = new SymfonyStyle($input, $output);
40          $locationID = $input->getArgument( name: 'id');
41          $location = $this->locationRepository->find($locationID);
42
43          $measurements = $this->weatherUtil->getWeatherForLocation($location);
44
45          $io->writeln(sprintf( format: 'Location: %s', $location->getCity()));
46          foreach ($measurements as $measurement) {
47              $io->writeln(sprintf( format: "\t%s: %s",
48                  $measurement->getDate()->format( format: 'Y-m-d'),
49                  $measurement->getCelsius()
50              ));
51          }
52
53          return Command::SUCCESS;
54      }
55  }

```

Wstaw zrzuty ekranu wyniku działania komendy dla dwóch lokalizacji:

```
PS C:\AI2-lab-v2\pogodynka> php bin\console weather:location 1
Location: Szczecin
      2025-01-24: 5
      2025-01-24: 3
      2025-01-25: 6
PS C:\AI2-lab-v2\pogodynka>
```

```
PS C:\AI2-lab-v2\pogodynka> php bin\console weather:location 2
Location: Police
      2025-01-23: 3
      2025-01-24: 6
PS C:\AI2-lab-v2\pogodynka>
```

Punkty:

0

1

W analogiczny sposób utwórz komendę do pobierania lokalizacji na podstawie kodu kraju i nazwy miejscowości.

Wstaw zrzuty ekranu kodu całości komendy:

```

WeatherLocationCommand.php  WeatherCityCommand.php x
15  #[AsCommand(
16      name: 'weather:city',
17      description: 'Add a short description for your command',
18  )]
19  class WeatherCityCommand extends Command
20  {
21      public function __construct(
22          private readonly LocationRepository $locationRepository,
23          private readonly WeatherUtil $weatherUtil,
24          string $name = null,
25      )
26      {
27          parent::__construct($name);
28      }
29      protected function configure(): void
30      {
31          $this
32              ->addArgument( name: 'country', mode: InputArgument::REQUIRED, description: 'Country code')
33              ->addArgument( name: 'city', mode: InputArgument::REQUIRED, description: 'City name')
34      };
35  }
36
37      protected function execute(InputInterface $input, OutputInterface $output): int
38      {
39          $io = new SymfonyStyle($input, $output);
40          $country = $input->getArgument( name: 'country');
41          $city = $input->getArgument( name: 'city');
42
43          $location = $this->locationRepository->findByCityAndCountry($city, $country);
44
45          $measurements = $this->weatherUtil->getWeatherForLocation($location);
46
47          $io->writeln(sprintf( format: 'Location: %s, %s', $location->getCity(), $location->getCountry()));
48          foreach ($measurements as $measurement) {
49              $io->writeln(sprintf( format: "\t%s: %s°C",
50                  $measurement->getDate()->format( format: 'Y-m-d'),
51                  $measurement->getCelsius()
52              ));
53          }
54      }
55      return Command::SUCCESS;

```

Wstaw zrzuty ekranu wyniku działania komendy dla dwóch miejscowości:

```
PS C:\AI2-lab-v2\pogodynka> php bin\console weather:city PL Szczecin
Location: Szczecin, PL
      2025-01-24: 5°C
      2025-01-24: 3°C
      2025-01-25: 6°C
PS C:\AI2-lab-v2\pogodynka> 
```

```
PS C:\AI2-lab-v2\pogodynka> php bin\console weather:city PL Police
Location: Police, PL
      2025-01-23: 3°C
      2025-01-24: 6°C
PS C:\AI2-lab-v2\pogodynka> 
```

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

COMMIT PROJEKTU DO GIT

Zacommituj zmiany. Wyślij zmiany do repozytorium (push). Upewnij się, czy wszystko dobrze się wysłało. Jeśli tak, to z poziomu przeglądarki utwórz branch o nazwie `lab-f` na podstawie głównej gałęzi kodu.

Podaj link do brancha `lab-f` w swoim repozytorium:

<https://github.com/AleksandraBancewicz/AI2/tree/main/LF>

PODSUMOWANIE

W kilku zdaniach podsumuj zdobyte podczas tego laboratorium umiejętności.

Implementacja komend w Symfony pozwala na wygodne wykonywanie zadań w aplikacji z poziomu terminala, co jest szczególnie przydatne przy operacjach na danych lub automatyzacji procesów. Dzięki dobrze zorganizowanej architekturze, jak serwisy (WeatherUtil) i repozytoria (LocationRepository), można w prosty sposób odseparować logikę biznesową od interfejsu użytkownika. Precyzyjne definiowanie argumentów w komendach pozwala na elastyczność i możliwość łatwego rozszerzenia funkcjonalności aplikacji. Proces tworzenia i dostosowywania komend pokazuje, jak ważne jest stosowanie dobrych praktyk w programowaniu, takich jak wykorzystanie wstrzykiwania zależności.

Zweryfikuj kompletność sprawozdania. Utwórz PDF i wyślij w terminie.