| AI2<br>LAB C | Zielaskowski Dawid<br>Album 51641<br>Grupa 2 | **Wersja 1** |
|---|---|---|

# BAZA DANYCH I OPERACJE ODCZYTU

## SPIS TREŚCI



## CEL ZAJĘĆ

Celem głównym zajęć jest zdobycie umiejętności tworzenia encji na podstawie diagramów ERD oraz opanowanie procesu tworzenia akcji w systemie monolitycznym – routing, kontroler, widok.

## ROZPOCZĘCIE

Rozpoczęcie zajęć. Powtórzenie zasad routingów w Symfony – atrybuty, adnotacje, yaml. Określanie parametrów. Określenie wymagań parametrów. Powtórzenie przekazywania parametrów do akcji kontrolera (parametry, serwisy, type-hinting i argument resolving). Powtórzenie TWIG – trzy typy wąsów, filtry (np. join, raw), pętle.

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`.

## ENCJA LOCATION

Pracuj wspólnie z resztą grupy. Utworzymy wspólnie encję `Location` z wykorzystaniem komendy `make:entity`.

Otwórz projekt `I:\AI2-lab\pogodynka` w PhpStorm / VS Code. W pliku .env zmień bazę danych na SQLITE:

```
# .env
#...
DATABASE_URL="sqlite:///%kernel.project_dir%/var/data.db"
```

Ten wpis oznacza, że aplikacja będzie korzystać z bazy danych SQLite umieszczonej w pliku `I:\AI2-lab\pogodynka\var\data.db`.

Otwórz terminal. Wykonaj polecenia, w celu utworzenia encji Location. Prowadzący omówi proces na udostępnionym ekranie:

```
cd I:\AI2-lab\pogodynka
php bin\console make:entity

Class name of the entity to create or update (e.g. GentleKangaroo):
 > Location

 created: src/Entity/Location.php
 created: src/Repository/LocationRepository.php

 Entity generated! Now let's add some fields!
 You can always add more fields later manually or by re-running this command.

 New property name (press <return> to stop adding fields):
 > city

 Field type (enter ? to see all types) [string]:
 >

 Field length [255]:
 >

 Can this field be null in the database (nullable) (yes/no) [no]:
 >

 updated: src/Entity/Location.php

 Add another property? Enter the property name (or press <return> to stop adding
fields):
 > country

 Field type (enter ? to see all types) [string]:
 >

 Field length [255]:
 > 2

 Can this field be null in the database (nullable) (yes/no) [no]:
```

```
 >

 updated: src/Entity/Location.php

 Add another property? Enter the property name (or press <return> to stop adding
fields):
 > latitude

 Field type (enter ? to see all types) [string]:
 > decimal

 Precision (total number of digits stored: 100.00 would be 5) [10]:
 > 10

 Scale (number of decimals to store: 100.00 would be 2) [0]:
 > 7

 Can this field be null in the database (nullable) (yes/no) [no]:
 >

 updated: src/Entity/Location.php

 Add another property? Enter the property name (or press <return> to stop adding
fields):
 > longitude

 Field type (enter ? to see all types) [string]:
 > decimal

 Precision (total number of digits stored: 100.00 would be 5) [10]:
 > 10

 Scale (number of decimals to store: 100.00 would be 2) [0]:
 > 7

 Can this field be null in the database (nullable) (yes/no) [no]:
 >

 updated: src/Entity/Location.php

 Add another property? Enter the property name (or press <return> to stop adding
fields):
 >



  Success!


 Next: When you're ready, create a migration with php bin/console make:migration
```

Razem z grupą omówcie powstałe pliki `Location.php` i `LocationRepository.php`.

Na tym etapie model danych nie został jeszcze naniesiony na bazę danych. Wykonaj komendy:
`php bin\console doctrine:schema:update --dump-sql`
`php bin\console doctrine:schema:update --dump-sql --force`

Czym różni się --dump-sql od --force?

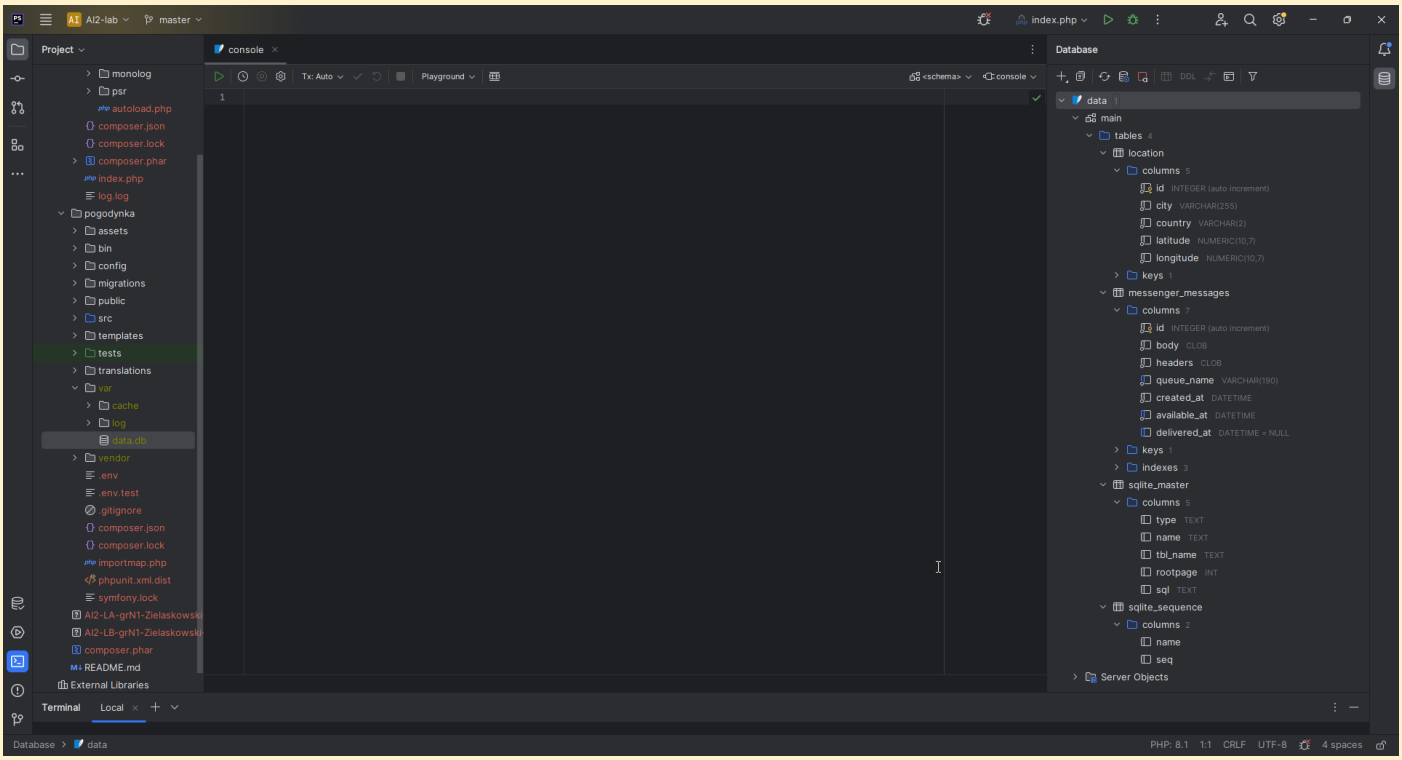
…omówienie różnicy…

--dump-sql wyświetla zapytania SQL bez ich wykonywania.
--force faktycznie wykonuje te zapytania na bazie danych, wprowadzając zmiany w schemacie.
Do sprawdzenia, co zostanie zmienione, użyj --dump-sql. Kiedy zmiany będą gotowe do wprowadzenia --force.

Umieść zrzut ekranu lub skopiuj SQL, który został wygenerowany:

CREATE TABLE location (id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, city VARCHAR(255) NOT NULL, country VARCHAR(2) NOT NULL, latitude NUMERIC(10, 7) NOT NULL, longitude NUMERIC(10, 7) NOT NULL);
CREATE TABLE messenger_messages (id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, body CLOB NOT NULL, headers CLOB NOT NULL, queue_name VARCHAR(190) NOT NULL, created_at DATETIME NOT NULL --(DC2Type:datetime_immutable)
, available_at DATETIME NOT NULL --(DC2Type:datetime_immutable)
, delivered_at DATETIME DEFAULT NULL --(DC2Type:datetime_immutable)
);
CREATE INDEX IDX_75EA56E0FB7336F0 ON messenger_messages (queue_name);
CREATE INDEX IDX_75EA56E0E3BD61CE ON messenger_messages (available_at);
CREATE INDEX IDX_75EA56E016BA31DB ON messenger_messages (delivered_at);

Wykorzystaj PhpStorm lub VS Code do połączenia się z bazą danych w pliku `I:\AI2-lab\pogodynka\var\data.db`. Umieść poniżej zrzut ekranu drzewa tabel/kolumn:



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

## POZOSTAŁE ENCJE

Stwórz pozostałe encje na podstawie swojego diagramu ERD z poprzednich zajęć. Zwrócić uwagę na typ danych `relation` przy tworzeniu relacji pomiędzy encjami.

Wymagane co najmniej encje `Location` i `Measurement` (lub odpowiedniki).

```
pogodynka> php .\bin\console make:entity

 Class name of the entity to create or update (e.g. DeliciousPopsicle):
 > Measurement

 created: src/Entity/Measurement.php
 created: src/Repository/MeasurementRepository.php

 Entity generated! Now let's add some fields!
 You can always add more fields later manually or by re-running this command.

 New property name (press <return> to stop adding fields):
 > location

 Field type (enter ? to see all types) [string]:
 > relation

 What class should this entity be related to?:
 > Location

What type of relationship is this?
 ------------ ---------------------------------------------------------------------
------
  Type         Description
 ------------ ---------------------------------------------------------------------
------
  ManyToOne    Each Measurement relates to (has) one Location.
               Each Location can relate to (can have) many Measurement objects.

  OneToMany    Each Measurement can relate to (can have) many Location objects.
               Each Location relates to (has) one Measurement.

  ManyToMany   Each Measurement can relate to (can have) many Location objects.
               Each Location can also relate to (can also have) many Measurement
objects.

  OneToOne     Each Measurement relates to (has) exactly one Location.
               Each Location also relates to (has) exactly one Measurement.
 ------------ ---------------------------------------------------------------------
------

 Relation type? [ManyToOne, OneToMany, ManyToMany, OneToOne]:
```

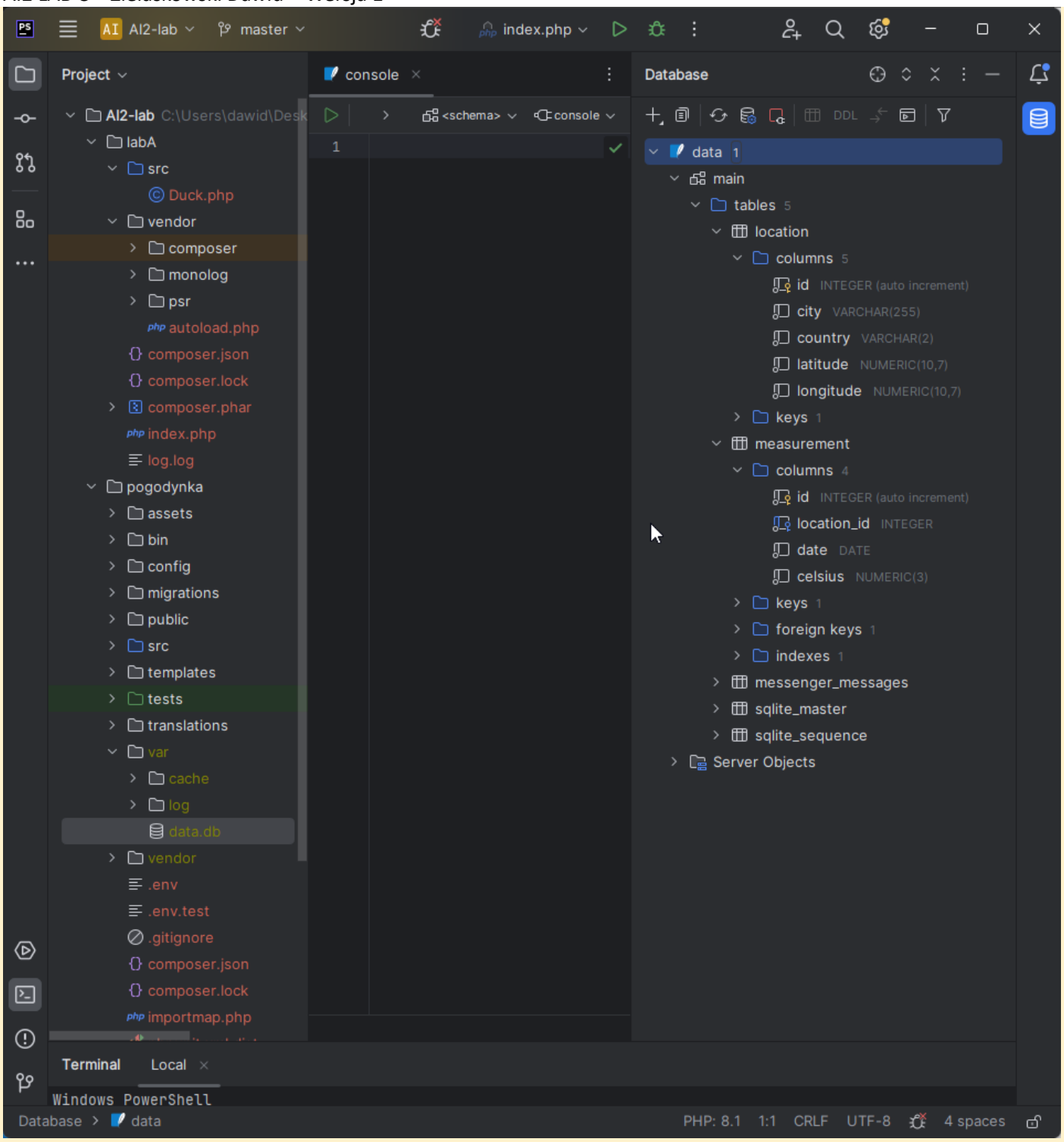
```
 > ManyToOne

 Is the Measurement.location property allowed to be null (nullable)? (yes/no) [yes]:
 > no

 Do you want to add a new property to Location so that you can access/update
Measurement objects from it - e.g. $location->getMeasurements()? (yes/no) [yes]:
 > yes

 A new property will also be added to the Location class so that you can access the
related Measurement objects from it.

 New field name inside Location [measurements]:
 >

 Do you want to activate orphanRemoval on your relationship?
 A Measurement is "orphaned" when it is removed from its related Location.
 e.g. $location->removeMeasurement($measurement)

 NOTE: If a Measurement may *change* from one Location to another, answer "no".

 Do you want to automatically delete orphaned App\Entity\Measurement objects
(orphanRemoval)? (yes/no) [no]:
 >

 updated: src/Entity/Measurement.php
 updated: src/Entity/Location.php

 Add another property? Enter the property name (or press <return> to stop adding
fields):
 > date

 Field type (enter ? to see all types) [string]:
 > date

 Can this field be null in the database (nullable) (yes/no) [no]:
 > no

 updated: src/Entity/Measurement.php

 Add another property? Enter the property name (or press <return> to stop adding
fields):
 > celsius

 Field type (enter ? to see all types) [string]:
 > decimal

 Precision (total number of digits stored: 100.00 would be 5) [10]:
 > 3

 Scale (number of decimals to store: 100.00 would be 2) [0]:
 > 0
```

```
 Can this field be null in the database (nullable) (yes/no) [no]:
 >

 updated: src/Entity/Measurement.php

 Add another property? Enter the property name (or press <return> to stop adding
fields):
 >



  Success!
```

Zsynchronizuj schemat bazy danych z utworzonymi encjami. Umieść poniżej wygenerowany i wykonany kod SQL:

CREATE TABLE measurement (id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, location_id INTEGER NOT NULL, date DATE NOT NULL, celsius NUMERIC(3, 0) NOT NULL, CONSTRAINT FK_2CE0D81164D218E FOREIGN KEY (location_id) REFERENCES location (id) NOT DEFERRABLE INITIALLY IMMEDIATE);
CREATE INDEX IDX_2CE0D81164D218E ON measurement (location_id);

Umieść poniżej zrzut ekranu podglądu zaktualizowanej bazy danych SQLite:




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

Finalnie, wypełnij bazę danych przykładowymi wpisami:

- Szczecin, PL, [53.4289, 14.553]
- Police, PL, [53.5521, 14.5718]

## KONTROLER

Utwórz pusty kontroler z wykorzystaniem komendy:

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

 Choose a name for your controller class (e.g. TinyPopsicleController):
 > WeatherController

 created: src/Controller/WeatherController.php
 created: templates/weather/index.html.twig


  Success!


 Next: Open your new controller class and add some pages!
```

Utworzony został plik `src/Controller/WeatherController.php`. Zwróć uwagę na wykorzystanie routingów w postaci atrybutów:

```php
<?php

namespace App\Controller;

use ...

no usages
class WeatherController extends AbstractController
{
    #[Route('/weather', name: 'app_weather')]
    public function index(): Response
    {
        return $this->render( view: 'weather/index.html.twig', [
            'controller_name' => 'WeatherController',
        ]);
    }
}
```

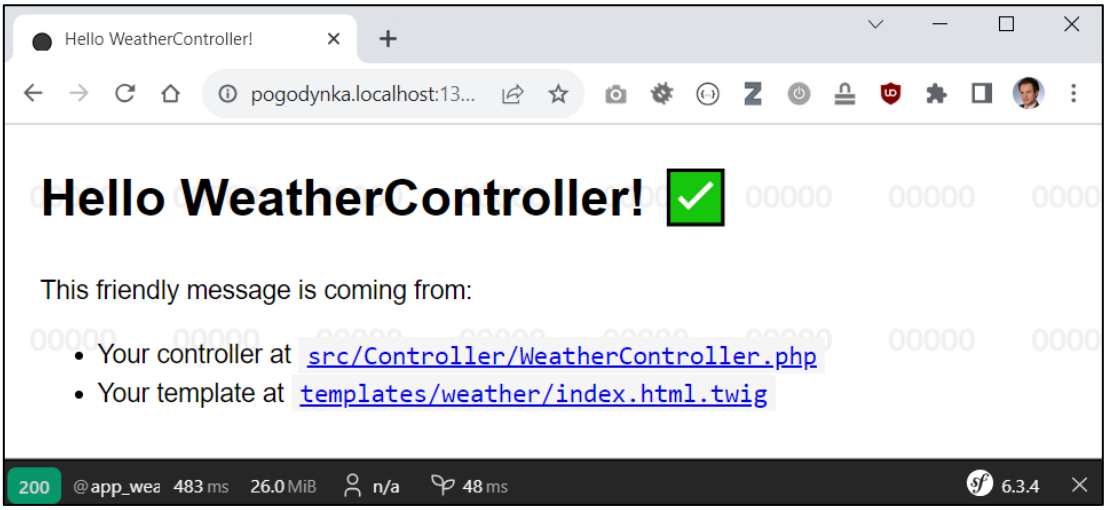
Utworzone zostały także pliki widoku:

- `templates/base.html.twig`
- `templates/weather/index.html.twig`

Zmodyfikuj plik `templates/base.html.twig` poprzez dodanie stylu w `<head>`, jako `text` wstawiając swój numer albumu:

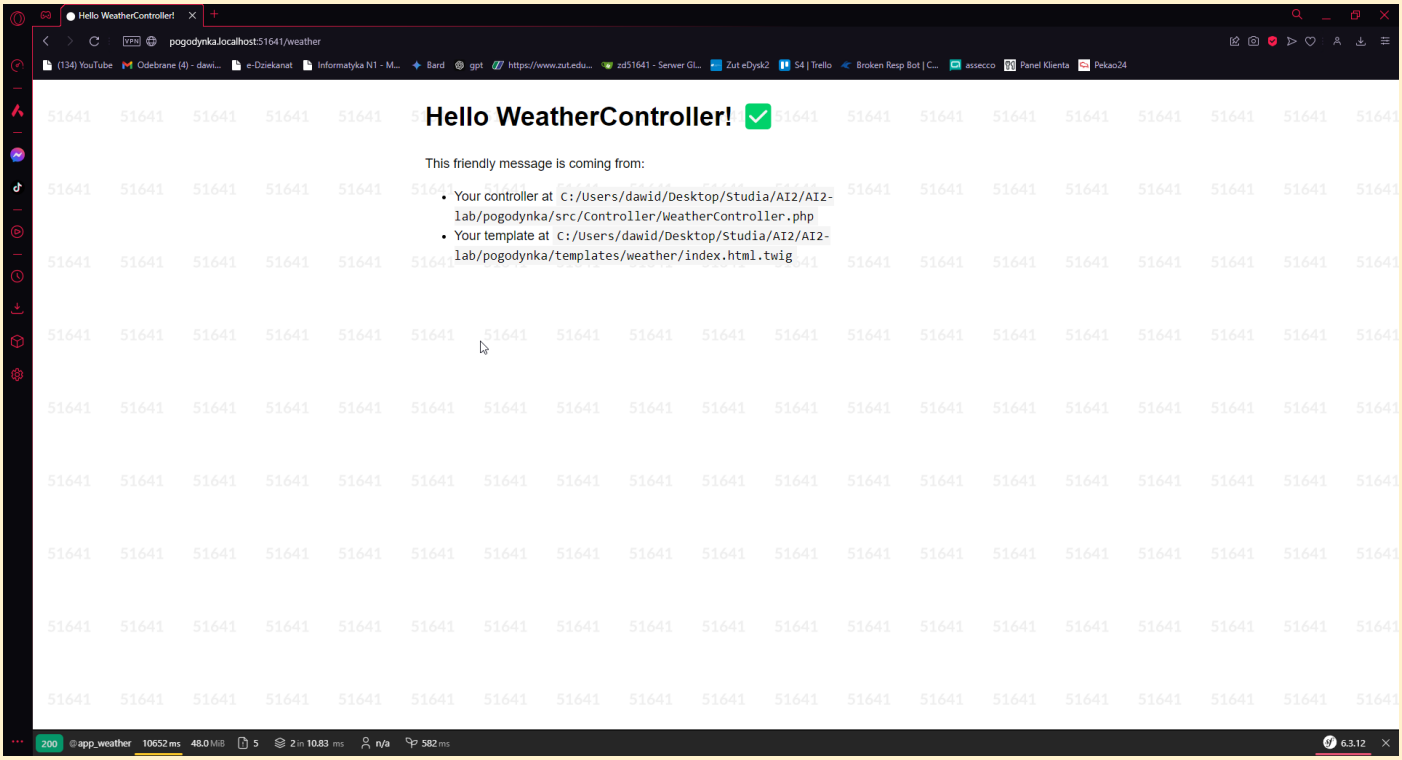
```
<style>
    body {
        background: url("https://placehold.co/100x100/FFFFFF/EFEFEF/png?text=51641");
    }
</style>
```

Akcję kontrolera można podejrzeć teraz w przeglądarce pod adresem sw:

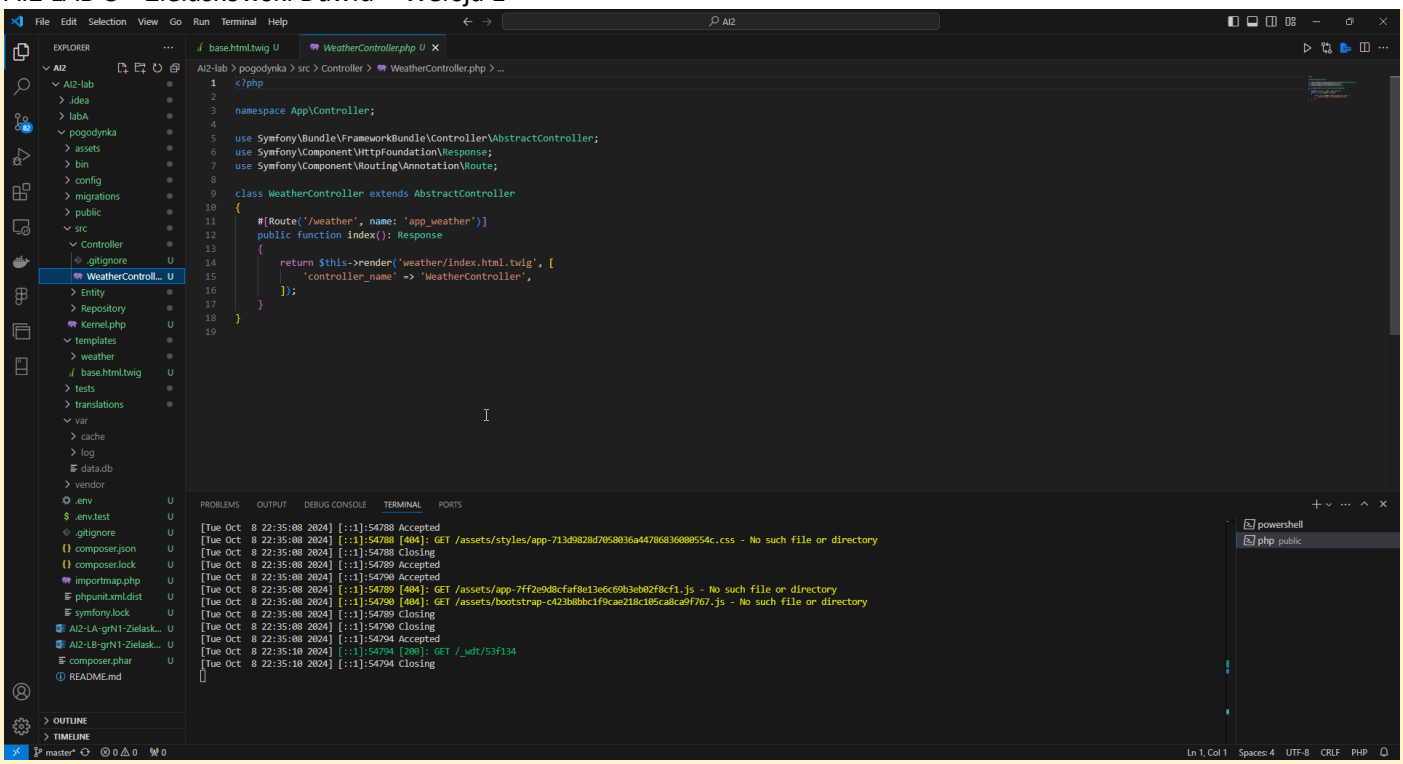



Na koniec zmień nazwę akcji kontrolera z `index` na `city`, a ścieżkę z `/weather` na `/weather/{id}`. Na ten moment wymuś, aby parametr `id` mógł być wyłącznie `\d+`.

Wstaw poniżej zrzut ekranu strony /weather/1, uwzględniający pasek adresu oraz tło z numerem indeksu:




Wstaw poniżej zrzut ekranu kodu kontrolera:




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

## REPOZYTORIUM

Zmodyfikujemy teraz naszą akcję w taki sposób, żeby pobierała dane. Otwórz w IDE plik src/Repository/MeasurementRepository.php i dodaj do niego metodę findByLocation:

```
public function findByLocation(Location $location)
{
    $qb = $this->createQueryBuilder('m');
    $qb->where('m.location = :location')
        ->setParameter('location', $location)
        ->andWhere('m.date > :now')
        ->setParameter('now', date('Y-m-d'));

    $query = $qb->getQuery();
    $result = $query->getResult();
    return $result;
}
```

Zmodyfikuj także kontroler, aby:

- automatycznie pobierał obiekt klasy Location na podstawie identyfikatora ze ścieżki URL;
- wykorzystywał metodę findByLocation do pobrania prognozy pogody dla zadanej lokacji;
- przekazywał informacje o lokacji i pobrane prognozy pogody na widok.

Przykładowo:

```
#[Route('/weather/{id}', name: 'app_weather', requirements: ['id' => '\d+'])]
public function city(Location $location, MeasurementRepository $repository): Response
{
    $measurements = $repository->findByLocation($location);

    return $this->render('weather/city.html.twig', [
        'location' => $location,
```

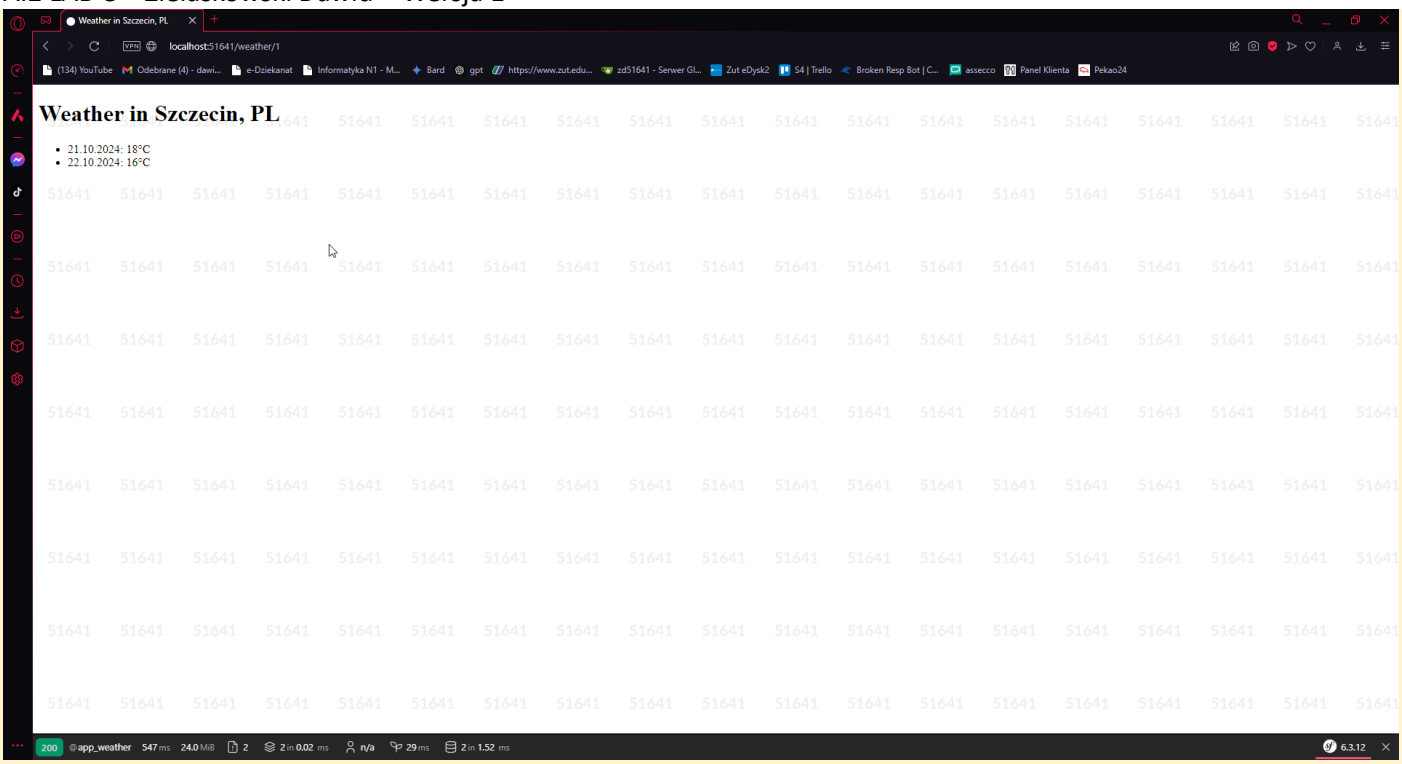
```
        'measurements' => $measurements,
    ]);
}
```

Na koniec edytuj widok (zmień `weather/index.html.twig` na `weather/city.html.twig`), aby wyświetlić informacje o lokacji i prognozę pogody:

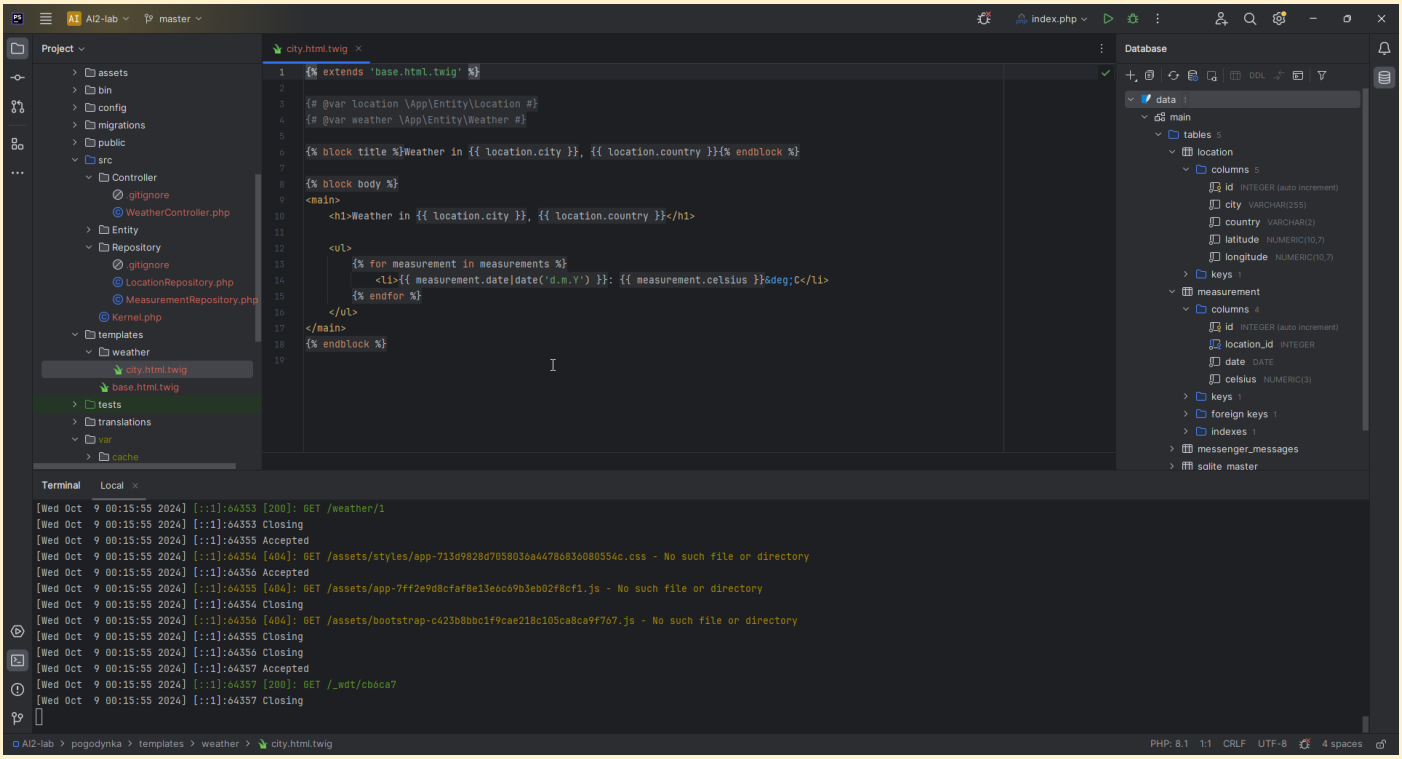
```
{% extends 'base.html.twig' %}

{# @var location \App\Entity\Location #}
{# @var weather \App\Entity\Weather #}

{% block title %}Weather in {{ location.city }}, {{ location.country }}{% endblock %}

{% block body %}
<main>
    <h1>Weather in {{ location.city }}, {{ location.country }}</h1>

    <ul>
        {% for measurement in measurements %}
            <li>{{ measurement.date|date('d.m.Y') }}: {{ measurement.celsius }}&deg;C</li>
        {% endfor %}
    </ul>
</main>
{% endblock %}
```

Oczekiwany efekt:


Wstaw zrzut ekranu wyglądu strony `/weather/...` z prognozą pogody dla pojedynczej lokacji:

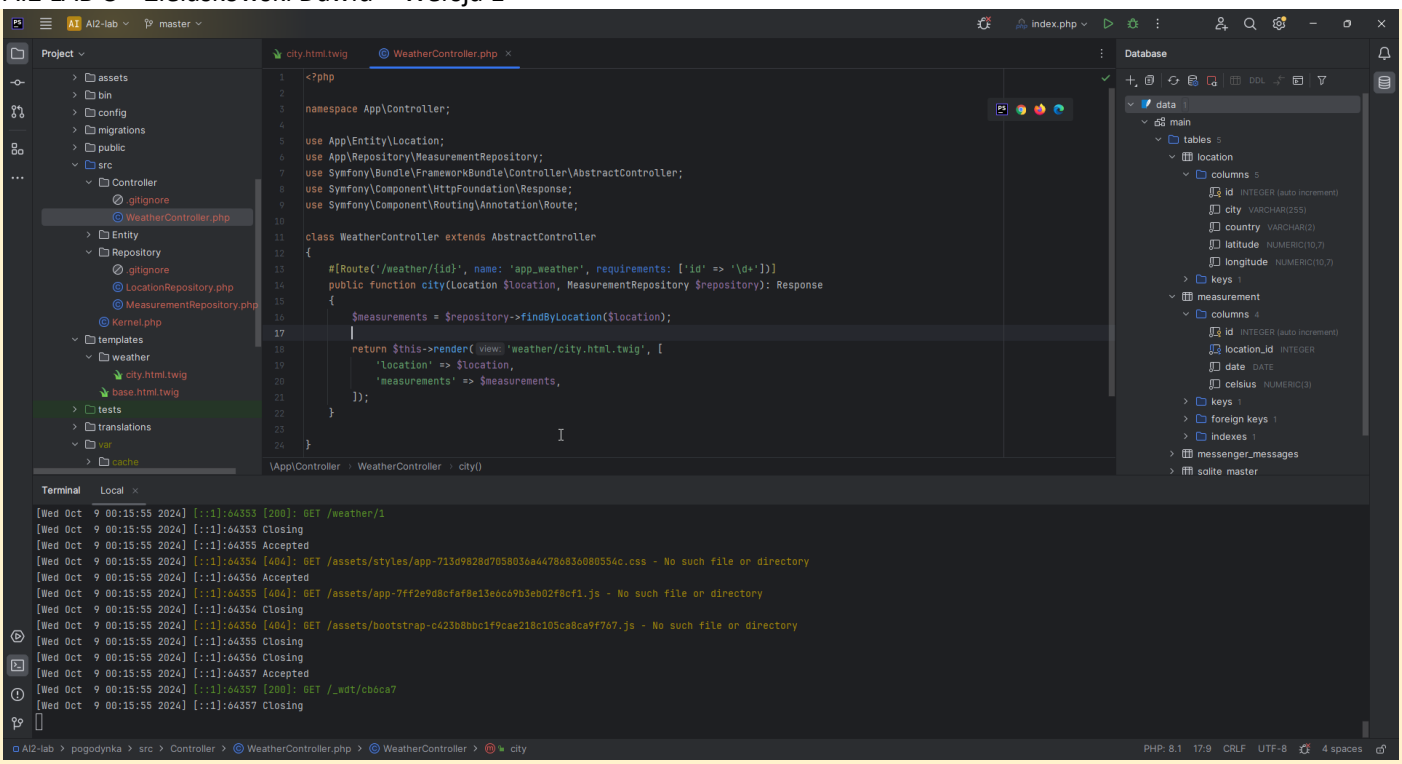



Wstaw zrzut ekranu kodu widoku `weather/city.html.twig`:

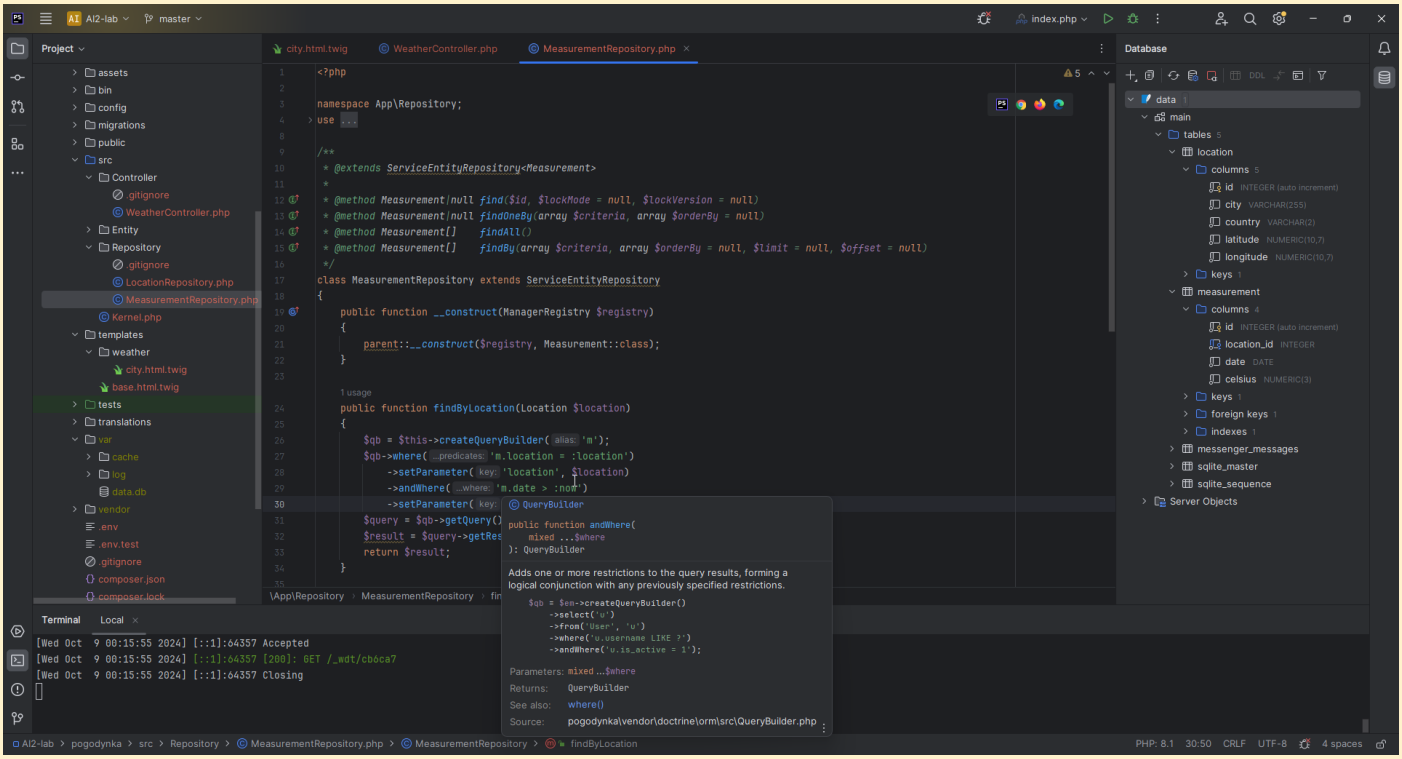



| Punkty: | 0 | 1 |
|---|---|---|

Wstaw zrzut ekranu kodu kontrolera `WeatherController`:




Wstaw zrzut ekranu kodu repozytorium `MeasurementRepository`:



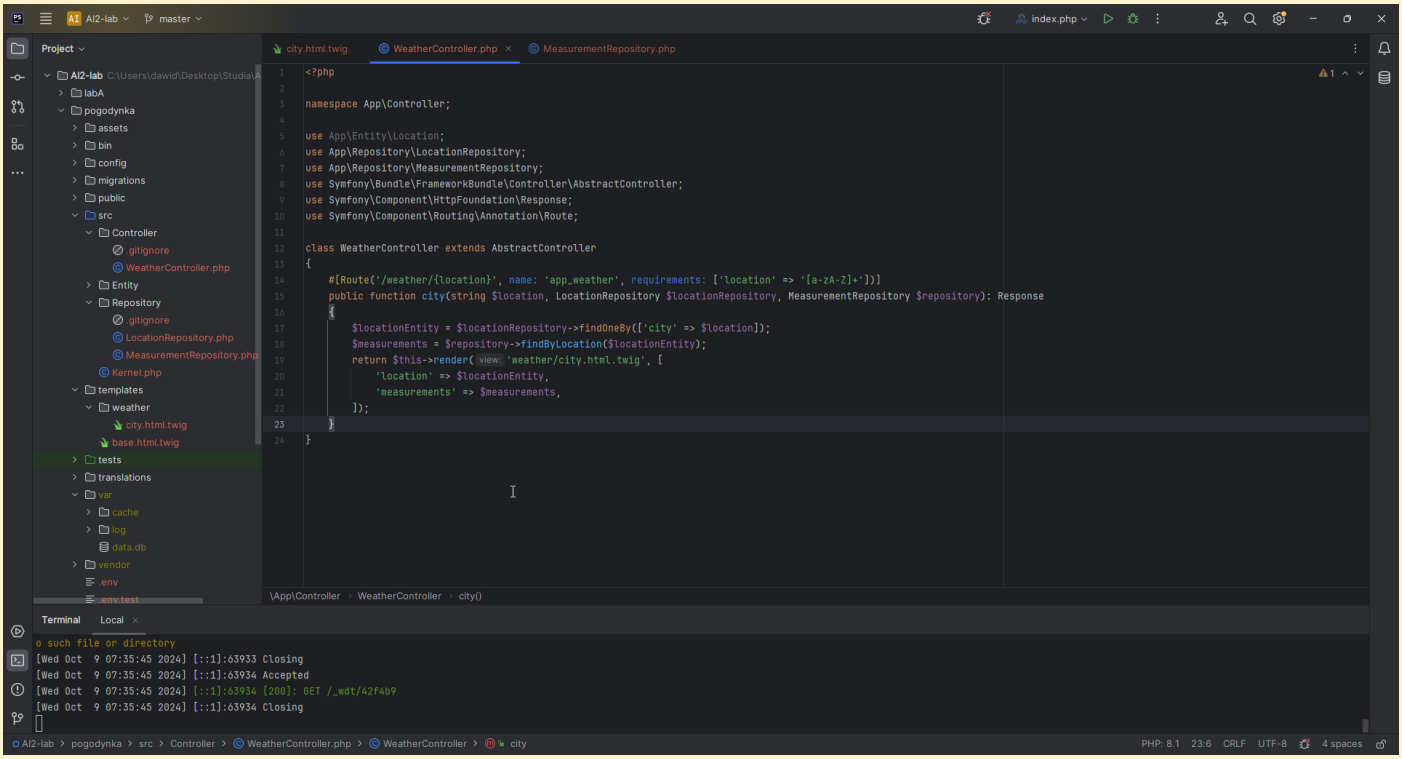

| Punkty: | 0 | 1 |
|---|---|---|

# WYSZUKIWANIE LOKACJI PO NAZWIE MIASTA

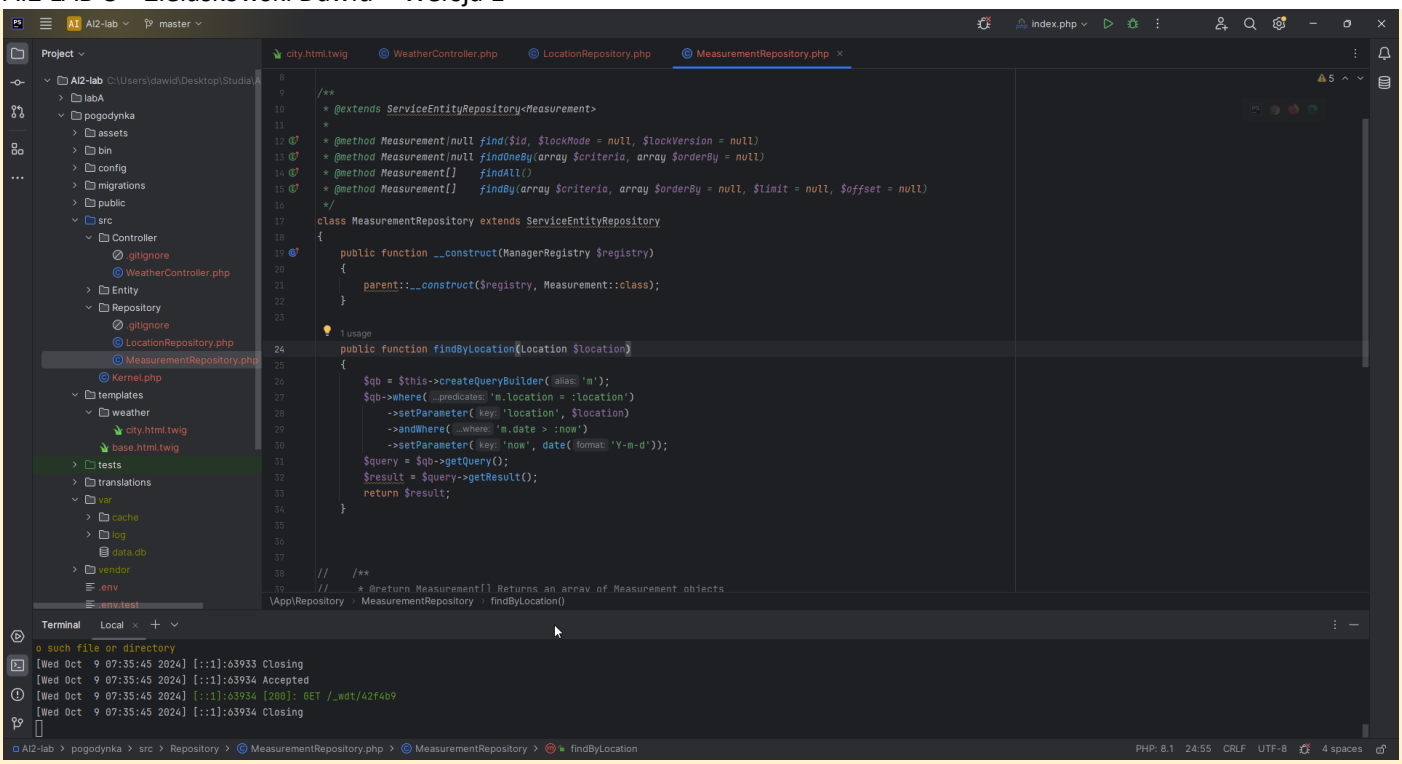
Zmodyfikuj kod akcji `WeatherController:city()` w taki sposób, żeby przyjmowała w ścieżce parametr z nazwą miejscowości (i opcjonalnie kodem państwa) zamiast parametru ID.

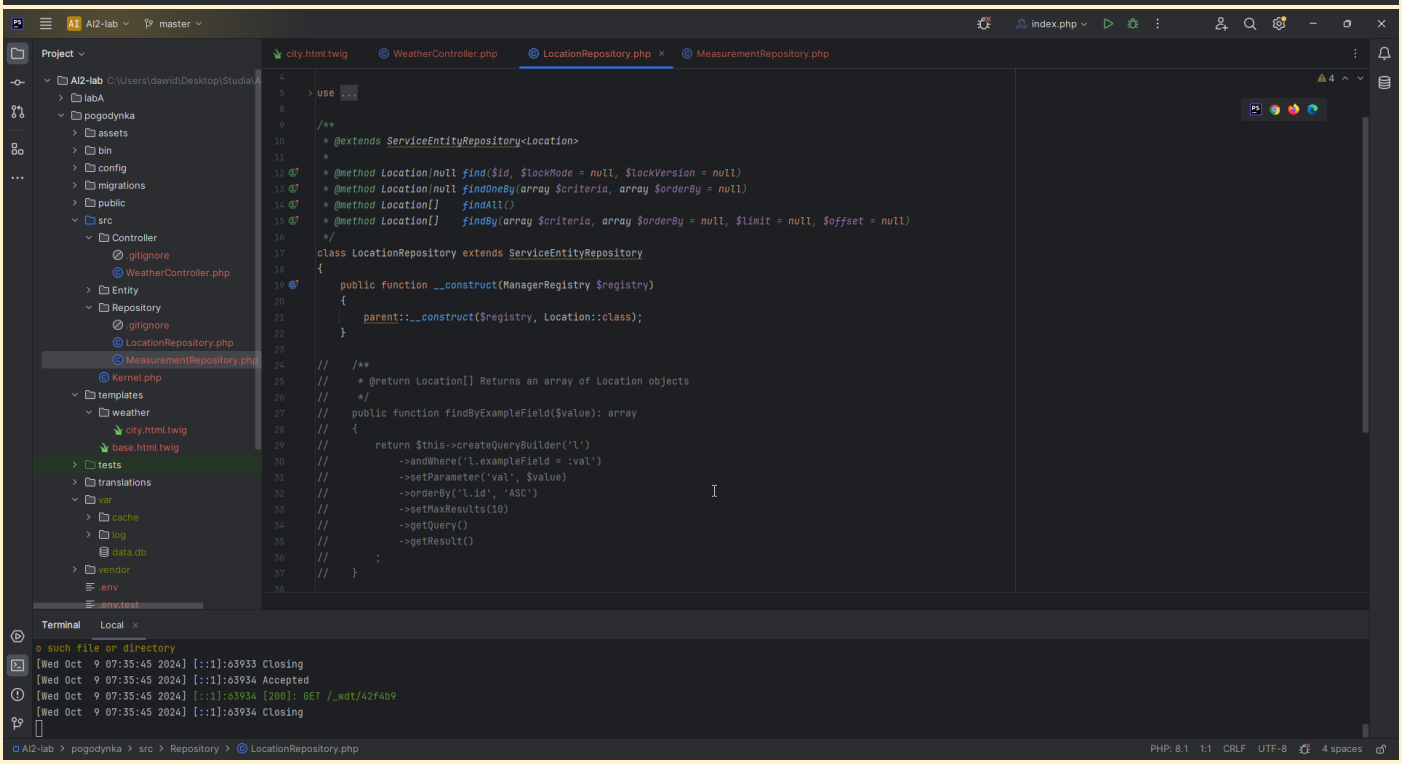
Warto poczytać: https://symfony.com/doc/current/doctrine.html#doctrine-entity-value-resolver.

Wstaw zrzut ekranu kodu zmodyfikowanego kontrolera:

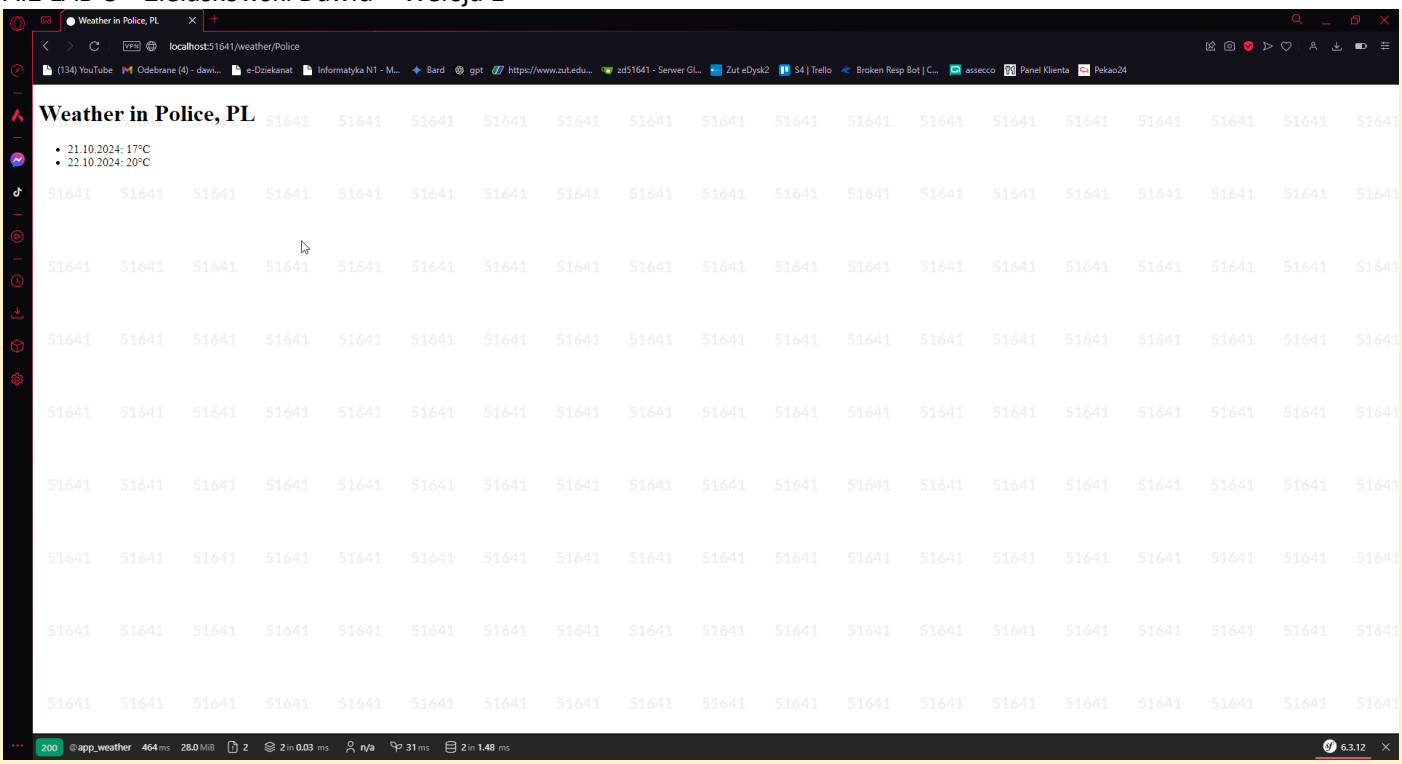



Wstaw zrzuty ekranu kodu zmodyfikowanych repozytoriów:




Wstaw zrzut ekranu wynikowej strony pod adresem uwzględniającym nazwę miasta:




| | | |
|---|---|---|
| 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-c` na podstawie głównej gałęzi kodu.

Podaj link do brancha `lab-c` w swoim repozytorium:
https://github.com/dawid-zielaskowski/ai2-pogodynka/tree/lab-c
…link, np. https://github.com/ideaspot-pl/ai2-pogodynka-202310/tree/lab-c…

## PODSUMOWANIE

W kilku zdaniach podsumuj zdobyte podczas tego laboratorium umiejętności.

…podsumowanie…

Zweryfikuj kompletność sprawozdania. Utwórz PDF i wyślij w terminie.