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

Baza danych i operacje odczytu

# Spis treści

[Spis treści 1](#_Toc180047557)

[Cel zajęć 1](#_Toc180047558)

[Rozpoczęcie 1](#_Toc180047559)

[Uwaga 1](#_Toc180047560)

[Encja Location 2](#_Toc180047561)

[Pozostałe encje 4](#_Toc180047562)

[Kontroler 7](#_Toc180047563)

[Repozytorium 10](#_Toc180047564)

[Wyszukiwanie lokacji po nazwie miasta 13](#_Toc180047565)

[Commit projektu do GIT 17](#_Toc180047566)

[Podsumowanie 17](#_Toc180047567)

# 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

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?

--dump-sql : Tylko wyświetla SQL, który zostanie wykonany w celu zaktualizowania bazy danych. Przydatne do sprawdzenia, jakie zapytania będą wykonane, bez ryzyka wprowadzenia niechcianych zmian.  
--force : Faktycznie wprowadza zmiany w bazie danych na podstawie schematu encji.   
Zaleca się najpierw użycie --dump-sql, aby sprawdzić, jakie zmiany zostaną wprowadzone, zanim faktycznie zdecydujemy się je wykonać za pomocą --force.

Umieść zrzut ekranu lub skopiuj SQL, który został wygenerowany:

Obraz zawierający zrzut ekranu, tekst

Opis wygenerowany automatycznie

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:

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

|  |  |  |
| --- | --- | --- |
| 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]:

>

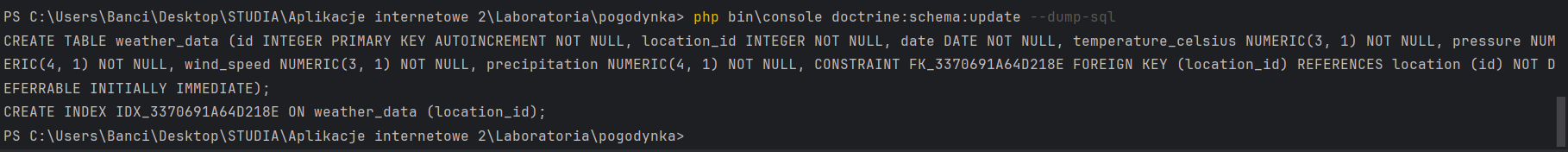
updated: src/Entity/Measurement.php

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

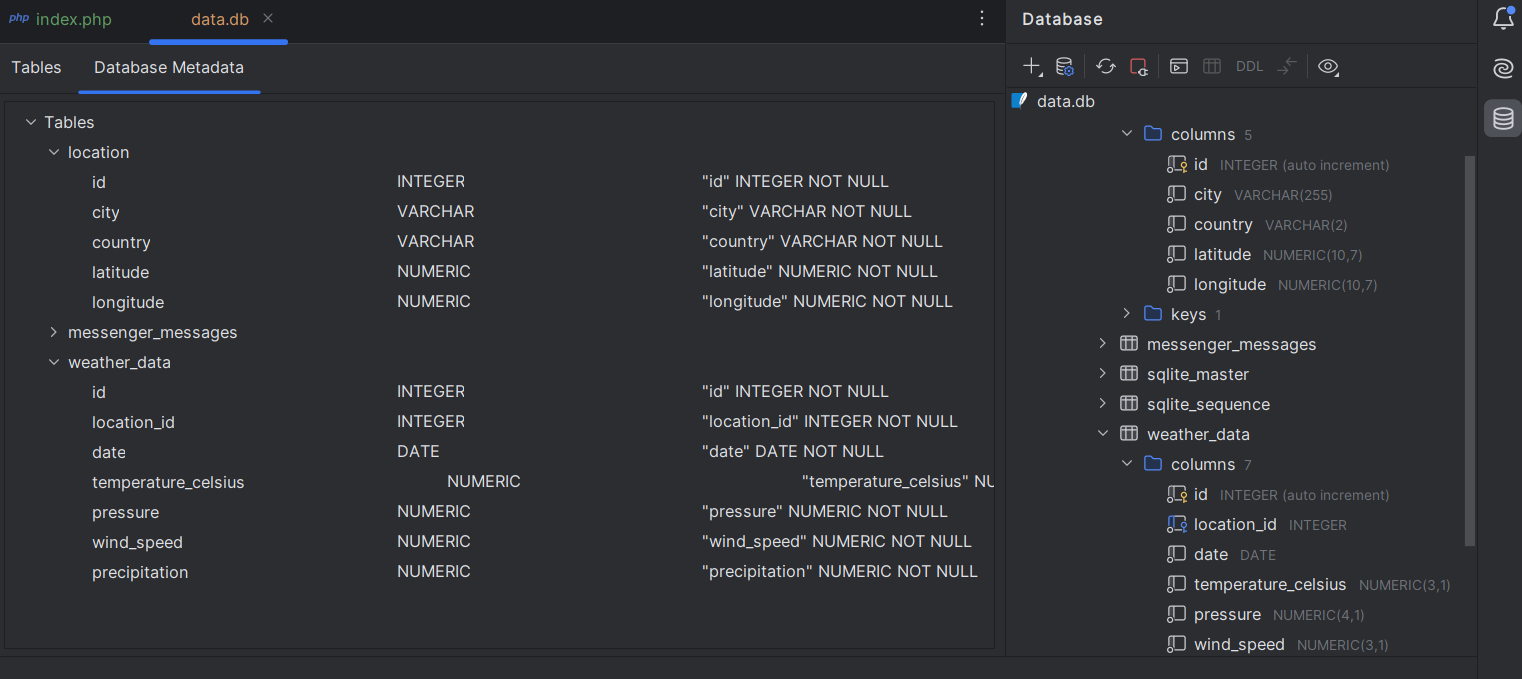
>

Success!

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



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:

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Strona internetowa

Opis wygenerowany automatycznie

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>

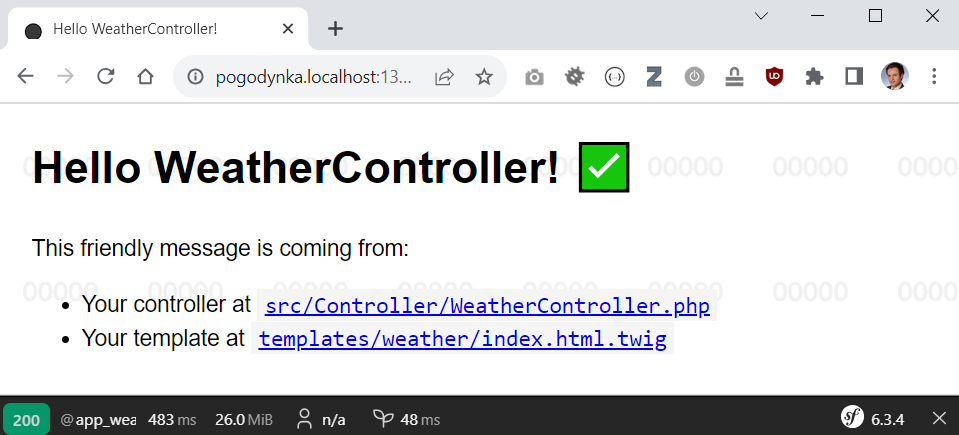
body {

background: url("https://placehold.co/100x100/FFFFFF/EFEFEF/png?text=ba34753");

}

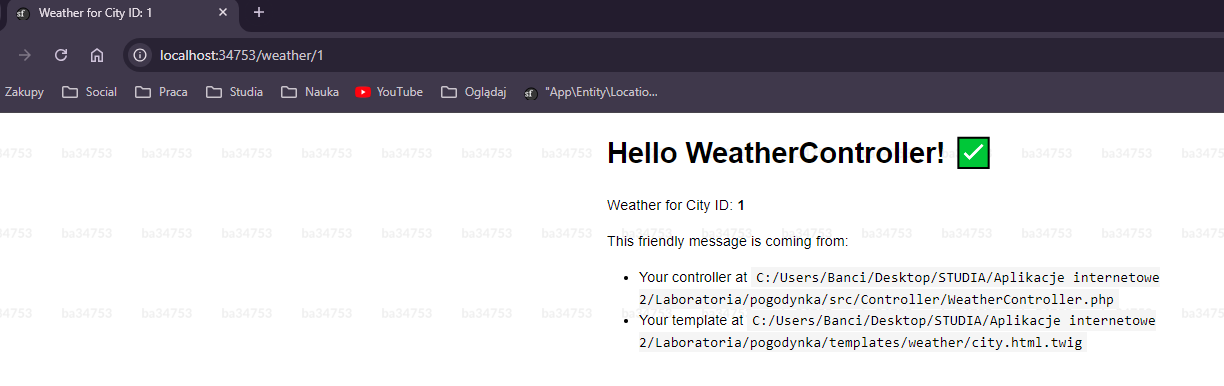
</style>

Akcję kontrolera można podejrzeć teraz w przeglądarce pod adresem http://pogodynka.localhost:ba34753/weather:



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:

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, wyświetlacz

Opis wygenerowany automatycznie

|  |  |  |
| --- | --- | --- |
| 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>

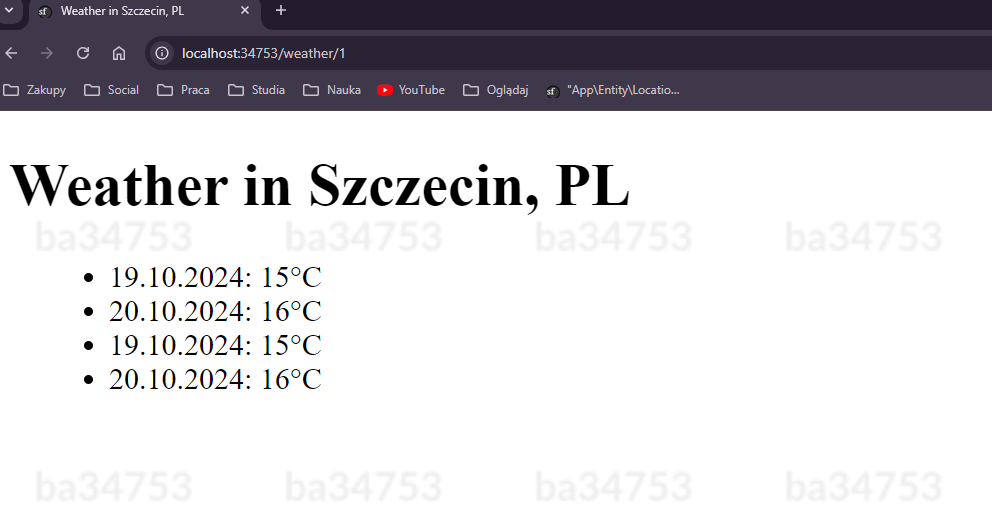
{% endblock %}

Oczekiwany efekt:

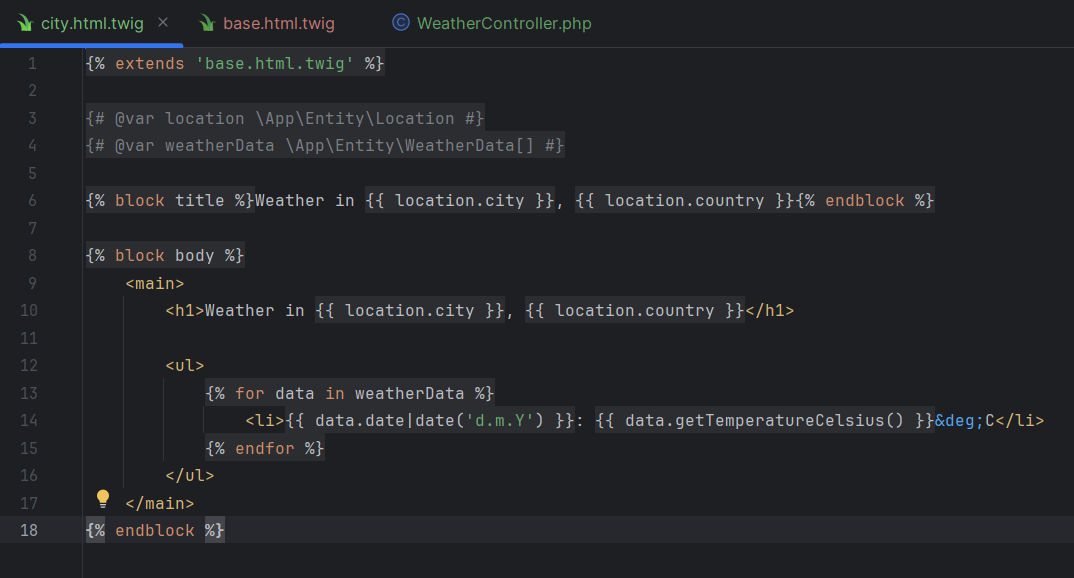
Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

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



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



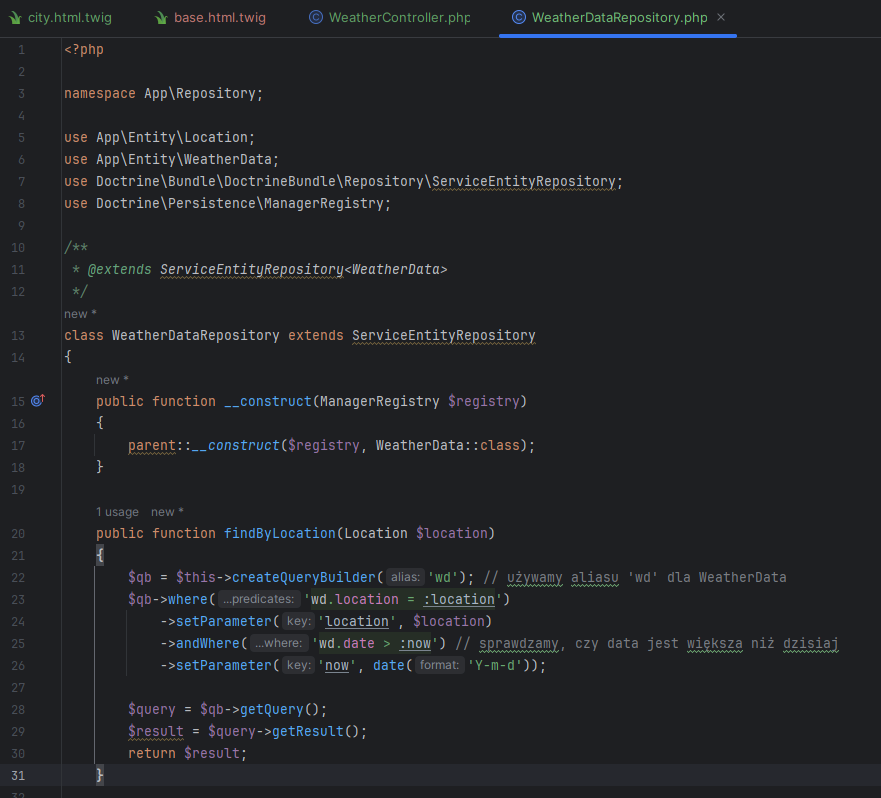
|  |  |  |
| --- | --- | --- |
| Punkty: | 0 | 1 |

Wstaw zrzut ekranu kodu kontrolera WeatherController:

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, wyświetlacz

Opis wygenerowany automatycznie

Wstaw zrzut ekranu kodu repozytorium MeasurementRepository:



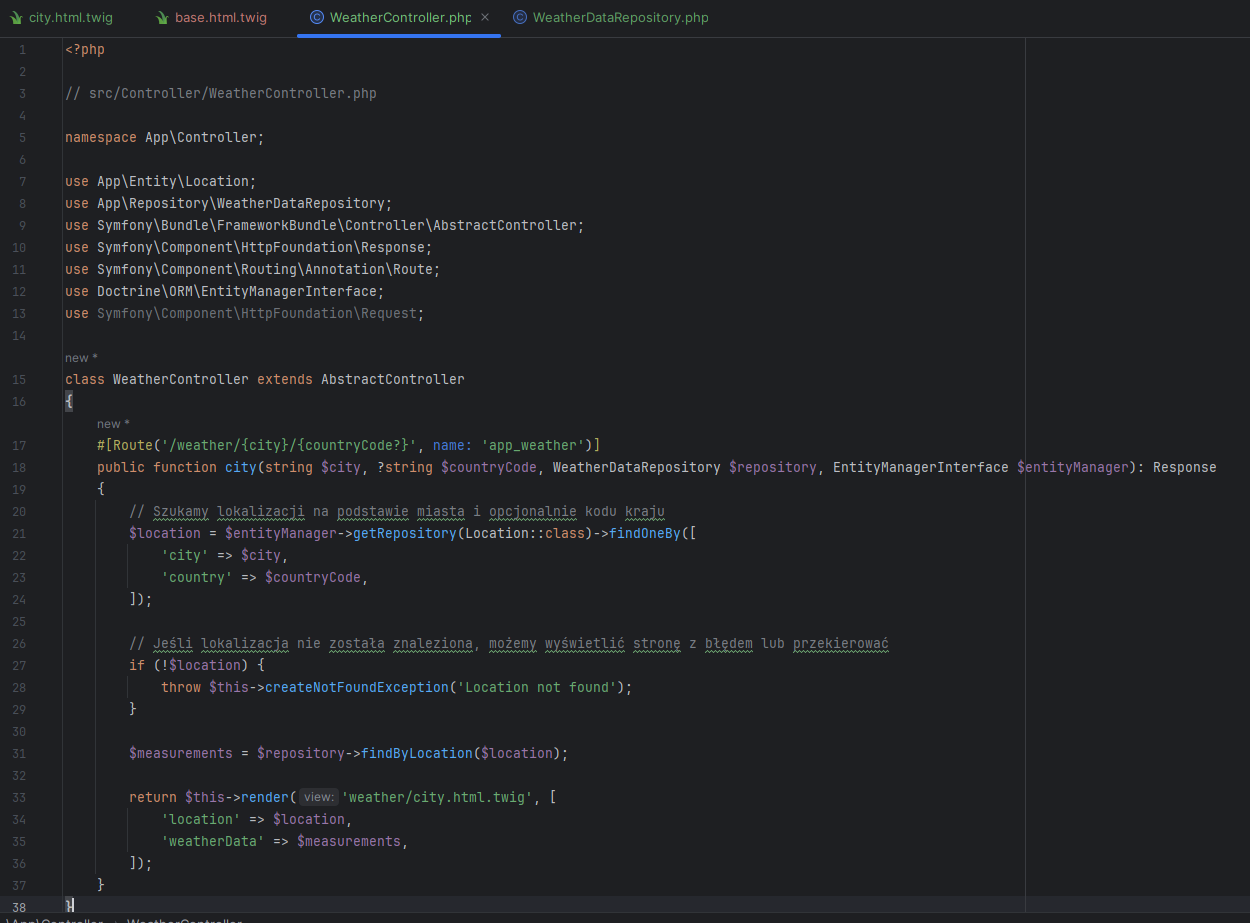
|  |  |  |
| --- | --- | --- |
| Punkty: | 0 | 1 |

# Wyszukiwanie lokacji po nazwie miasta

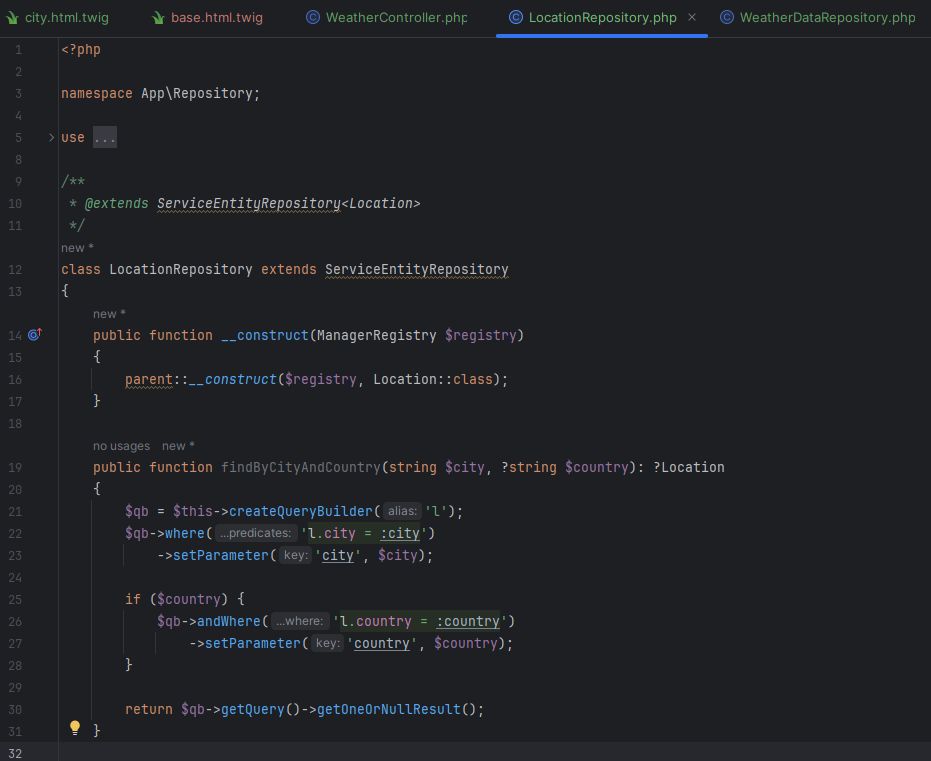
Zmodyfikuj kod akcji WeatherController:city() w taki sposób, żeby przyjmowała w ścieżce parametr z nazwą miejscowości (i opcjonalnie kodem państwa) zamiast parametru ID.

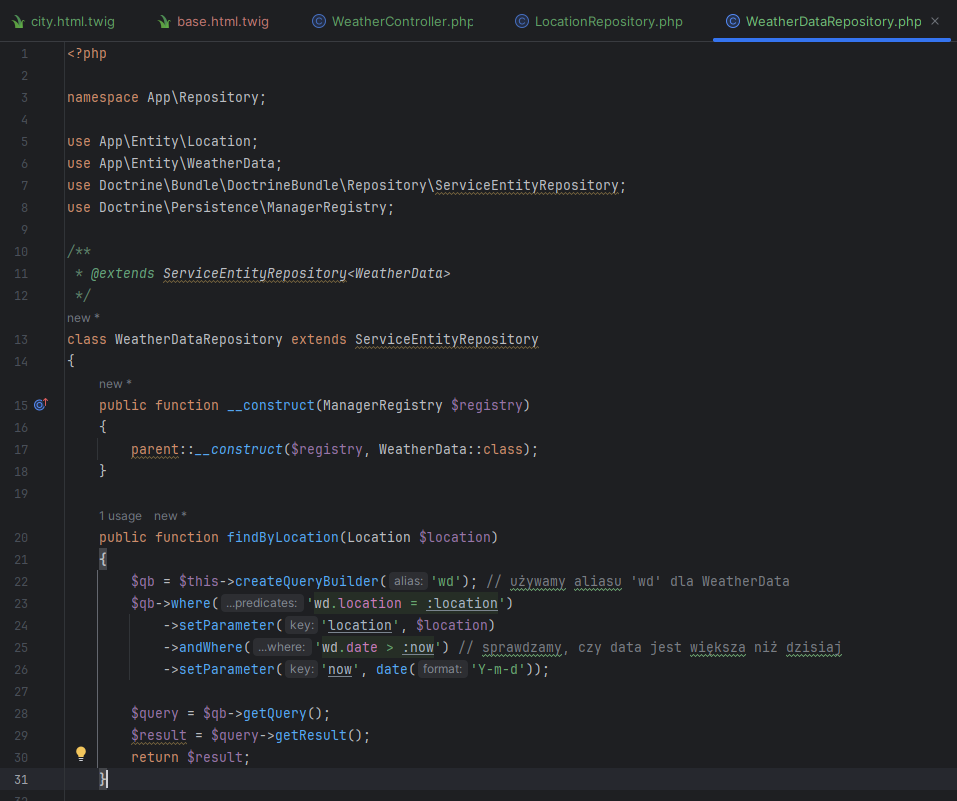
Warto poczytać: <https://symfony.com/doc/current/doctrine.html#doctrine-entity-value-resolver>.

Wstaw zrzut ekranu kodu zmodyfikowanego kontrolera:



Wstaw zrzuty ekranu kodu zmodyfikowanych repozytoriów:





Wstaw zrzut ekranu wynikowej strony pod adresem uwzględniającym nazwę miasta:

Obraz zawierający tekst, zrzut ekranu, Czcionka, oprogramowanie

Opis wygenerowany automatycznie

|  |  |  |
| --- | --- | --- |
| 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/AleksandraBancewicz/AI2/tree/main/LC

# Podsumowanie

W kilku zdaniach podsumuj zdobyte podczas tego laboratorium umiejętności.

Podczas tego laboratorium zdobyłam umiejętności związane z tworzeniem i modyfikowaniem aplikacji internetowych przy użyciu Symfony oraz Doctrine. Nauczyłam się, jak implementować funkcje do wyszukiwania danych w bazie danych, jak obsługiwać routing oraz jak dynamicznie przekazywać dane z kontrolerów do widoków. Ponadto zrozumiałam, jak używać Gita do zarządzania wersjami projektu, w tym jak dodawać zmiany do repozytoriów i tworzyć nowe gałęzie. Udoskonaliłam także umiejętności związane z pracą z bazami danych SQLite, w tym dodawanie danych i obsługę ograniczeń. Całość pozwoliła mi lepiej zrozumieć proces tworzenia aplikacji webowych i integracji z bazą danych

Zweryfikuj kompletność sprawozdania. Utwórz PDF i wyślij w terminie.