

BAZA DANYCH I OPERACJE ODCZYTU

SPIS TREŚCI

Spis treści	1
Cel zajęć.....	1
Rozpoczęcie	1
Uwaga	1
Encja Location	2
Pozostałe encje.....	4
Kontroler	7
Repozytorium.....	10
Wyszukiwanie lokacji po nazwie miasta.....	13
Commit projektu do GIT.....	17
Podsumowanie.....	17

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

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.

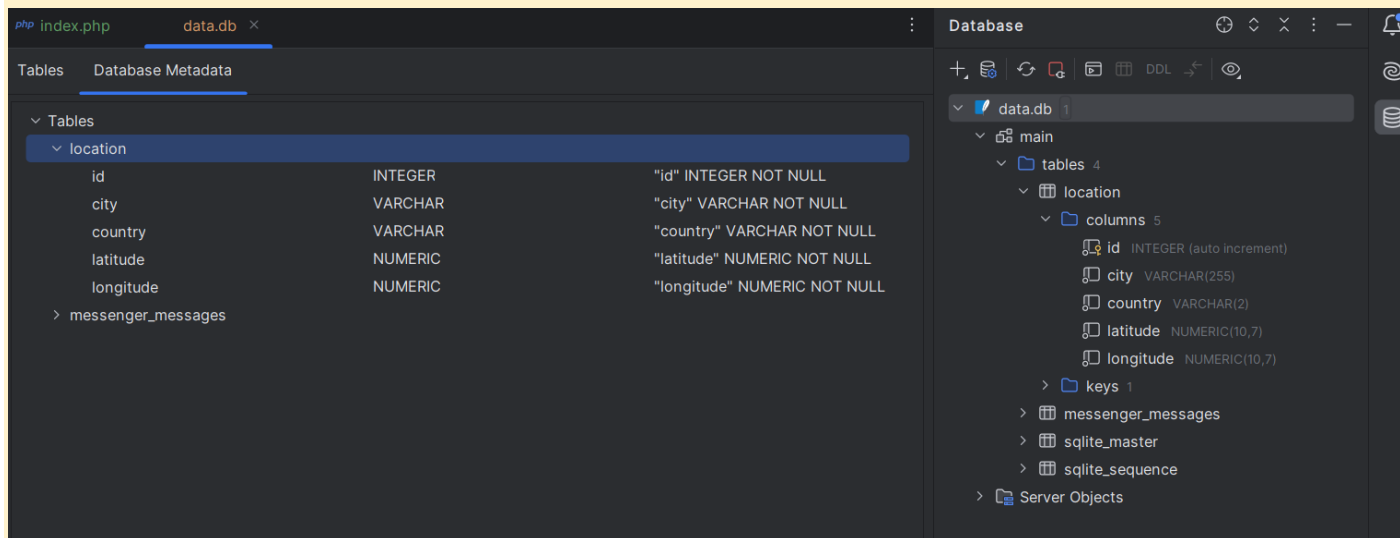
--force : Faktycznie wprowadza zmiany w bazie danych na podstawie schematu encji.

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:

```
PS C:\Users\Banci\Desktop\STUDIA\Aplikacje internetowe 2\Laboratoria\pogodynka> php bin\console doctrine:schema:update --dump-sql
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);
DROP TABLE doctrine_migration_versions;
DROP TABLE weather_data;
PS C:\Users\Banci\Desktop\STUDIA\Aplikacje internetowe 2\Laboratoria\pogodynka>
```

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. Zwróć uwagę na typ danych `relation` przy tworzeniu relacji pomiędzy encjami.

```
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:

```
PS C:\Users\Banci\Desktop\STUDIA\Aplikacje internetowe 2\Laboratoria\pogodynka> php bin\console doctrine:schema:update --dump-sql
CREATE TABLE weather_data (id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, location_id INTEGER NOT NULL, date DATE NOT NULL, temperature_celsius NUMERIC(3, 1) NOT NULL, pressure NUMERIC(4, 1) NOT NULL, wind_speed NUMERIC(3, 1) NOT NULL, precipitation NUMERIC(4, 1) NOT NULL, CONSTRAINT FK_3370691A64D218E FOREIGN KEY (location_id) REFERENCES location (id) NOT DEFERRABLE INITIALLY IMMEDIATE);
CREATE INDEX IDX_3370691A64D218E ON weather_data (location_id);
PS C:\Users\Banci\Desktop\STUDIA\Aplikacje internetowe 2\Laboratoria\pogodynka>
```

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:

```

1  <?php
2
3  namespace App\Controller;
4
5  > use ...
6
7
8
9  no usages
10 class WeatherController extends AbstractController
11 {
12     #[Route('/weather', name: 'app_weather')]
13     public function index(): Response
14     {
15         return $this->render( view: 'weather/index.html.twig', [
16             'controller_name' => 'WeatherController',
17         ]);
18     }
19 }
```

Utworzone zostały także pliki widoku:

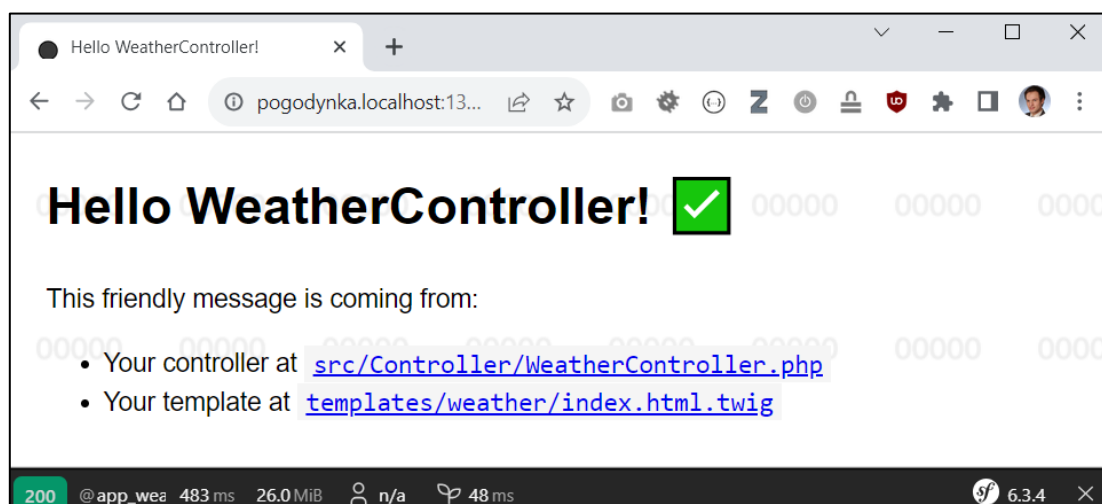
- `templates/base.html.twig`
- `templates/weather/index.html.twig`

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

```

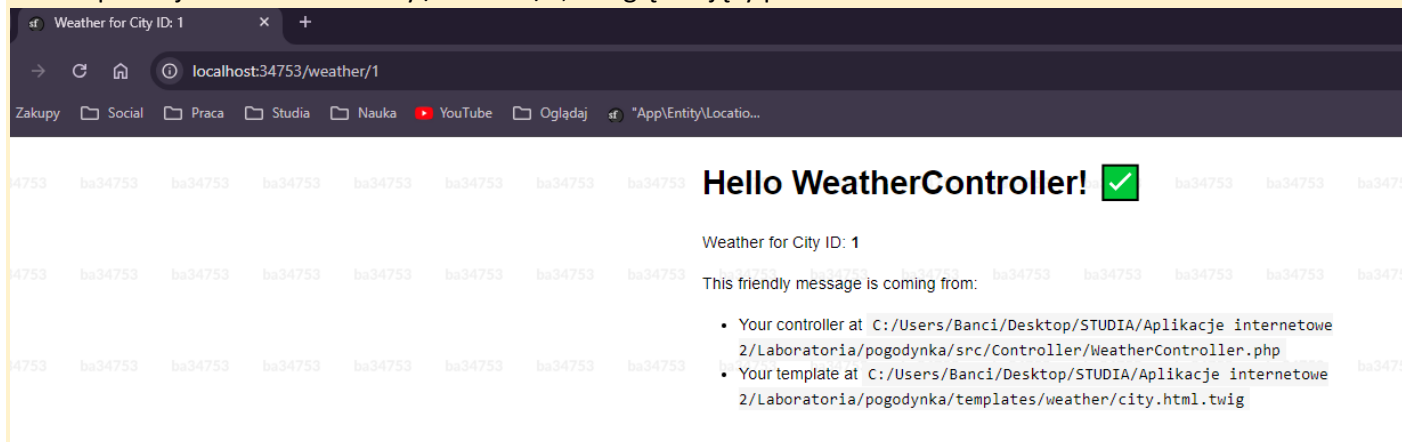
<style>
  body {
    background: url("https://placeholder.co/100x100/FFFFFF/EFEFEF/png?text=XXXXXX");
  }
</style>
```

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

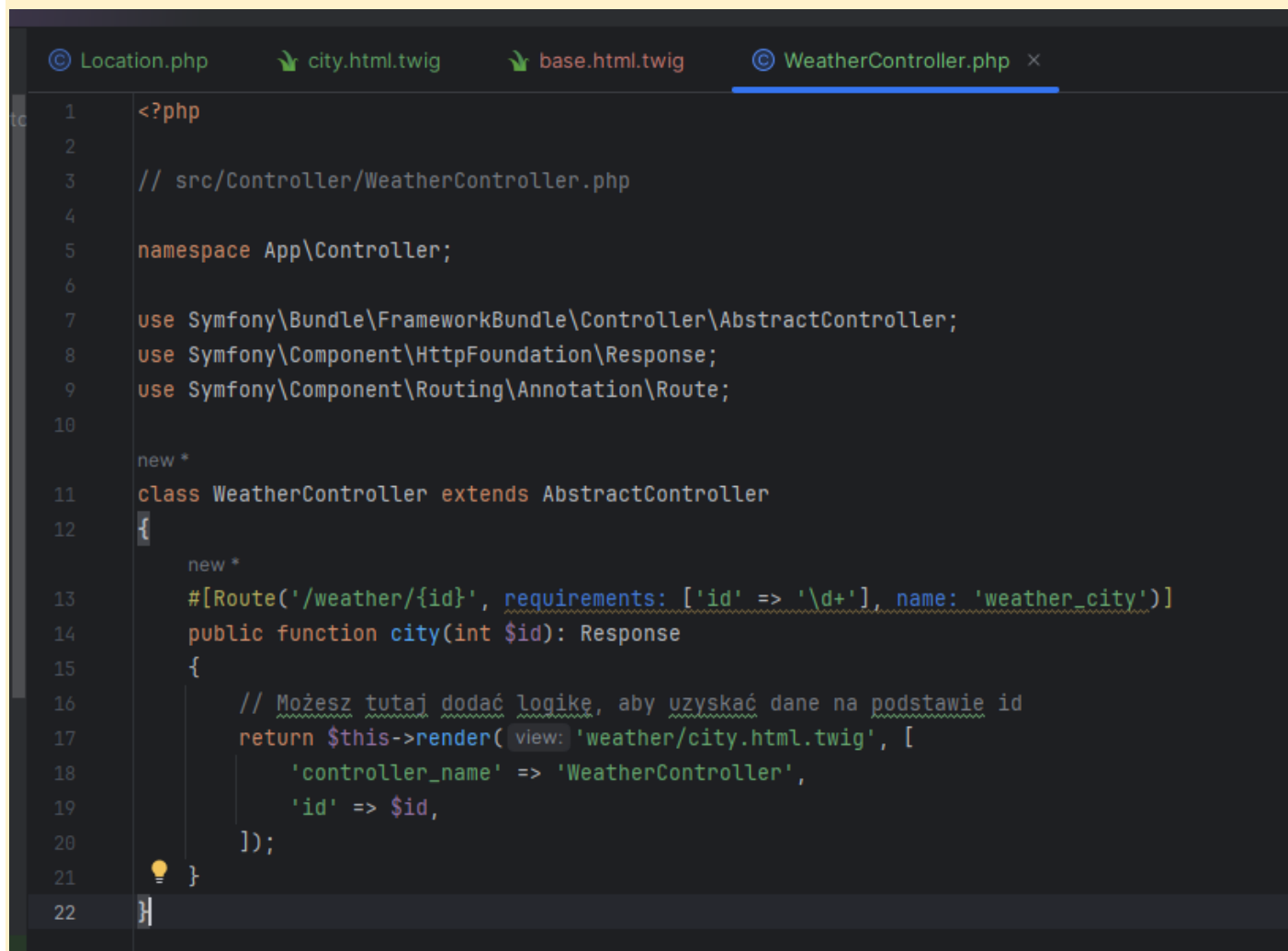


Na koniec zmień nazwę akcji kontrolera z `index` na `city`, a ścieżkę z `/weather` na `/weather/{id}`. Na ten moment wymuś, aby parametr `id` mógł być wyłącznie `\d+`.

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



Wstaw poniżej zrzut ekranu kodu kontrolera:



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 %}

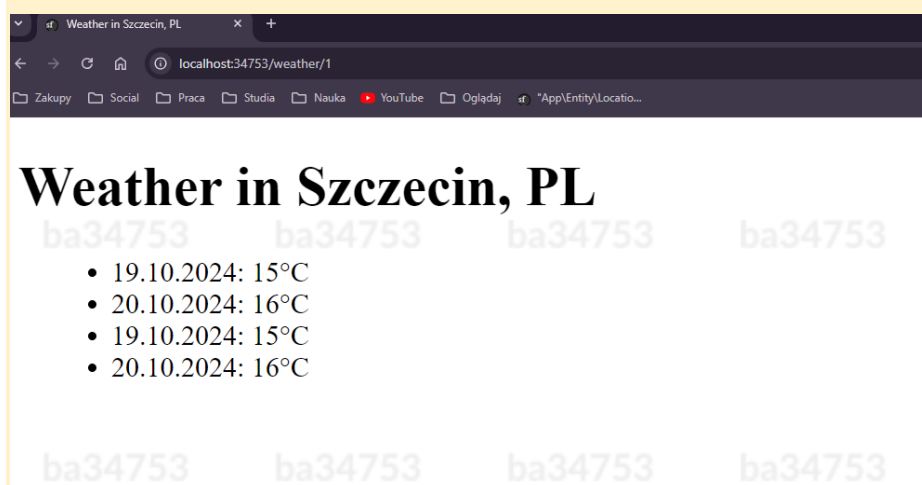
{% block body %}
<main>
    <h1>Weather in {{ location.city }}, {{ location.country }}</h1>

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

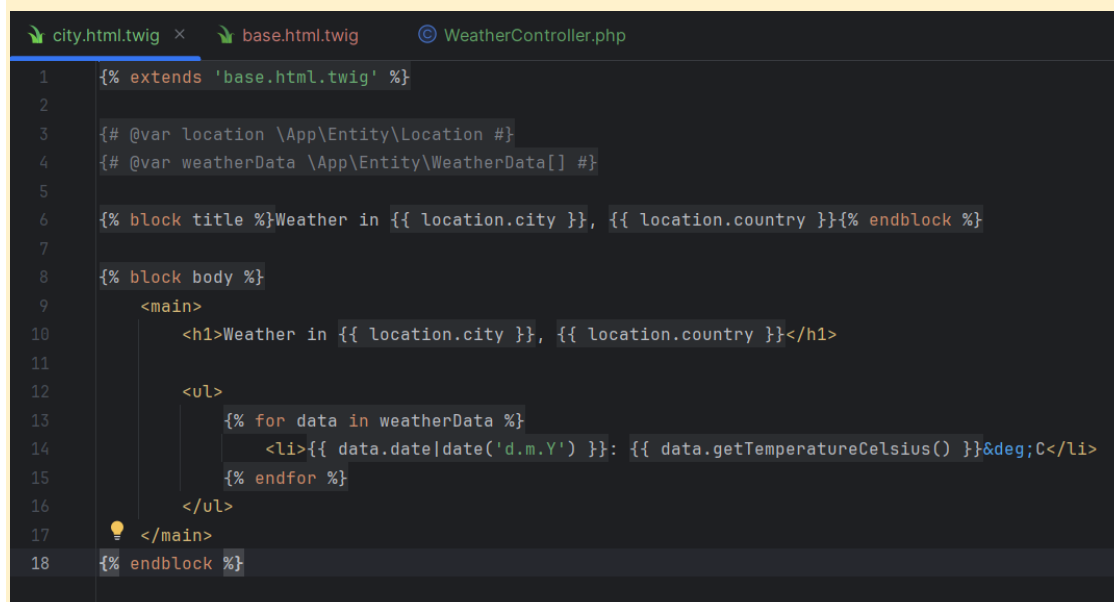
Oczekiwany efekt:



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:

```
city.html.twig base.html.twig © WeatherController.php x
1  <?php
2
3  namespace App\Controller;
4
5  use App\Entity\Location;
6  use App\Repository\WeatherDataRepository;
7  use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
8  use Symfony\Component\HttpFoundation\Response;
9  use Symfony\Component\Routing\Annotation\Route;
10
11  new *
12  class WeatherController extends AbstractController
13  {
14      new *
15      #[Route('/weather/{id}', name: 'app_weather', requirements: ['id' => '\d+'])]
16      public function city(Location $location, WeatherDataRepository $repository): Response
17      {
18          $weatherData = $repository->findByLocation($location);
19
20          return $this->render('weather/city.html.twig', [
21              'location' => $location,
22              'weatherData' => $weatherData,
23          ]);
24      }
25  }
```

Wstaw zrzut ekranu kodu repozytorium MeasurementRepository:

```

1  <?php
2
3  namespace App\Repository;
4
5  use App\Entity\Location;
6  use App\Entity\WeatherData;
7  use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
8  use Doctrine\Persistence\ManagerRegistry;
9
10 /**
11  * @extends ServiceEntityRepository<WeatherData>
12  */
13 new *
14 class WeatherDataRepository extends ServiceEntityRepository
15 {
16     new *
17     public function __construct(ManagerRegistry $registry)
18     {
19         parent::__construct($registry, WeatherData::class);
20     }
21
22     1 usage new *
23     public function findByLocation(Location $location)
24     {
25         $qb = $this->createQueryBuilder(alias: 'wd'); // używamy aliasu 'wd' dla WeatherData
26         $qb->where(...predicates: 'wd.location = :location')
27             ->setParameter(key: 'location', $location)
28             ->andWhere(...where: 'wd.date > :now') // sprawdzamy, czy data jest większa niż dzisiaj
29             ->setParameter(key: 'now', date(format: 'Y-m-d'));
30
31         $query = $qb->getQuery();
32         $result = $query->getResult();
33         return $result;
34     }
35 }

```

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

WYSZUKIWANIE LOKACJI PO NAZWIE MIASTA

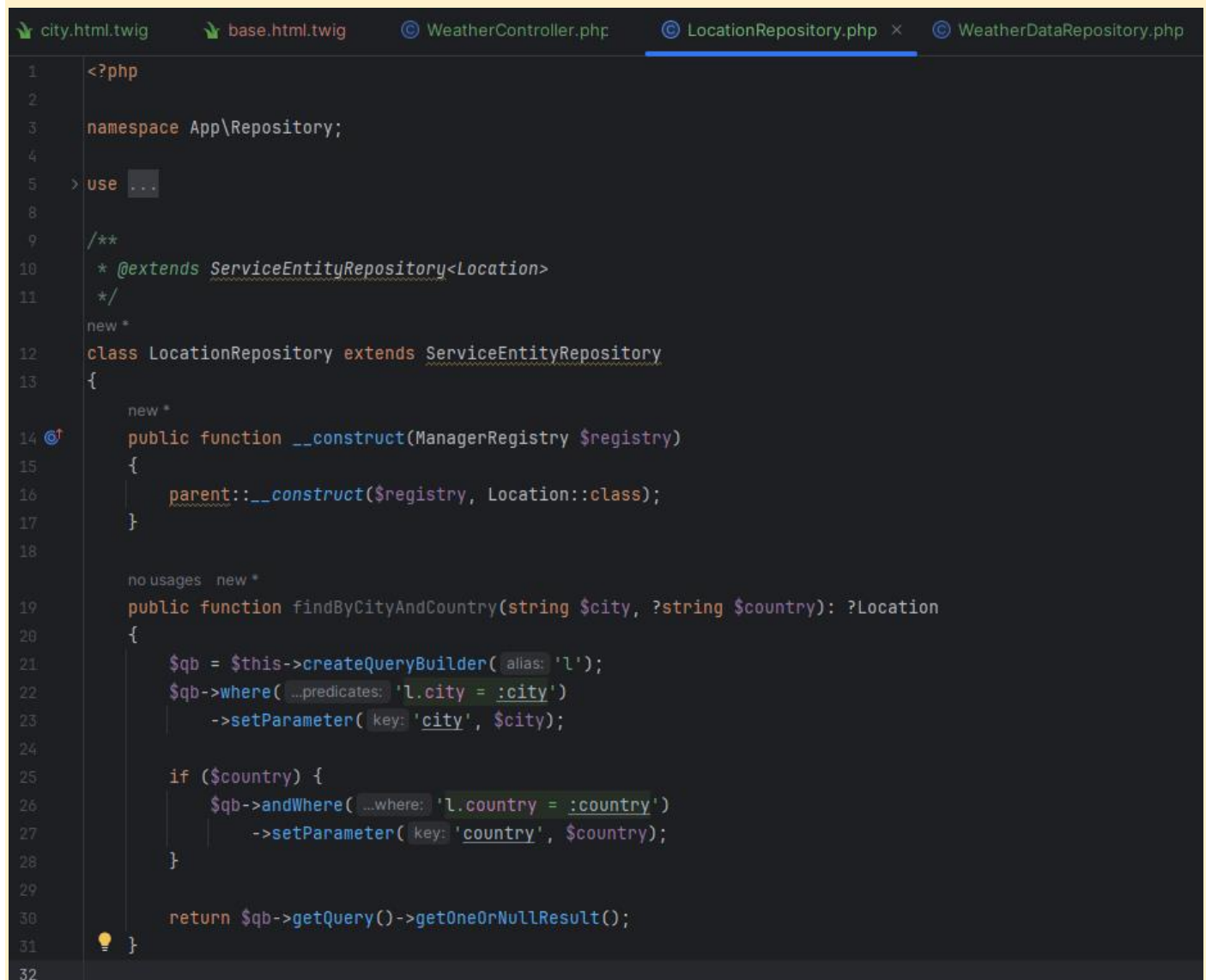
Zmodyfikuj kod akcji WeatherController:city() w taki sposób, żeby przyjmowała w ścieżce parametr z nazwą miejscowości (i opcjonalnie kodem państwa) zamiast parametru ID.

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

Wstaw zrzut ekranu kodu zmodyfikowanego kontrolera:

```
city.html.twig base.html.twig WeatherController.php x WeatherDataRepository.php
1 <?php
2
3 // src/Controller/WeatherController.php
4
5 namespace App\Controller;
6
7 use App\Entity\Location;
8 use App\Repository\WeatherDataRepository;
9 use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
10 use Symfony\Component\HttpFoundation\Response;
11 use Symfony\Component\Routing\Annotation\Route;
12 use Doctrine\ORM\EntityManagerInterface;
13 use Symfony\Component\HttpFoundation\Request;
14
15 new *
16 class WeatherController extends AbstractController
17 {
18     new *
19     #[Route('/weather/{city}/{countryCode?}', name: 'app_weather')]
20     public function city(string $city, ?string $countryCode, WeatherDataRepository $repository, EntityManagerInterface $entityManager): Response
21     {
22         // Szukamy lokalizacji na podstawie miasta i opcjonalnie kodu kraju
23         $location = $entityManager->getRepository(Location::class)->findOneBy([
24             'city' => $city,
25             'country' => $countryCode,
26         ]);
27
28         // Jeśli lokalizacja nie została znaleziona, możemy wyświetlić stronę z błędem lub przekierować
29         if (!$location) {
30             throw $this->createNotFoundException('Location not found');
31         }
32
33         $measurements = $repository->findByLocation($location);
34
35         return $this->render(view: 'weather/city.html.twig', [
36             'location' => $location,
37             'weatherData' => $measurements,
38         ]);
39     }
40 }
```

Wstaw zrzuty ekranu kodu zmodyfikowanych repozytoriów:



```

1  <?php
2
3  namespace App\Repository;
4
5  > use ...
6
7
8
9  /**
10   * @extends ServiceEntityRepository<Location>
11   */
12  new *
13  class LocationRepository extends ServiceEntityRepository
14  {
15      new *
16      public function __construct(ManagerRegistry $registry)
17      {
18          parent::__construct($registry, Location::class);
19      }
20
21      no usages new *
22      public function findByCityAndCountry(string $city, ?string $country): ?Location
23      {
24          $qb = $this->createQueryBuilder( alias: 'l');
25          $qb->where( ...predicates: 'l.city = :city')
26              ->setParameter( key: 'city', $city);
27
28          if ($country) {
29              $qb->andWhere( ...where: 'l.country = :country')
30                  ->setParameter( key: 'country', $country);
31          }
32
33          return $qb->getQuery()->getOneOrNullResult();
34      }
35  }

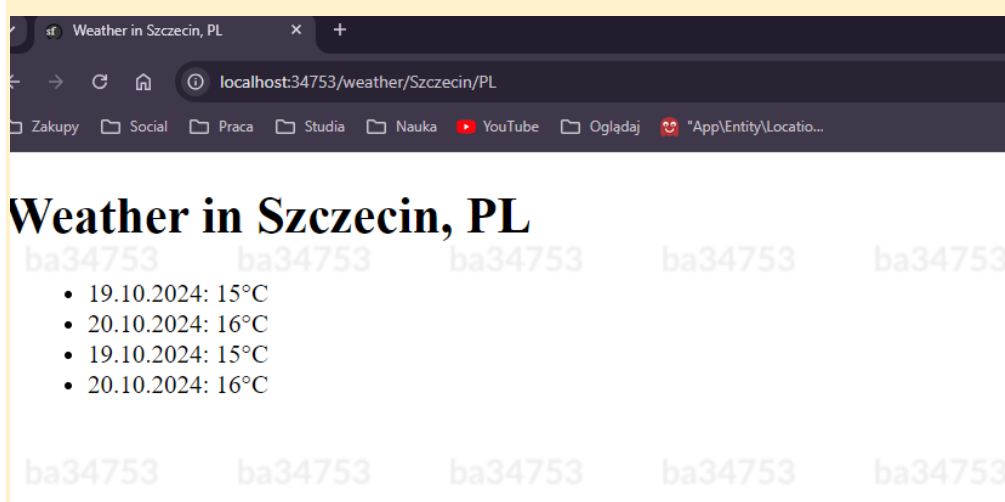
```

```

city.html.twig base.html.twig WeatherController.php LocationRepository.php WeatherDataRepository.php x
1 <?php
2
3 namespace App\Repository;
4
5 use App\Entity\Location;
6 use App\Entity\WeatherData;
7 use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
8 use Doctrine\Persistence\ManagerRegistry;
9
10 /**
11  * @extends ServiceEntityRepository<WeatherData>
12  */
13 new *
14 class WeatherDataRepository extends ServiceEntityRepository
15 {
16     new *
17     public function __construct(ManagerRegistry $registry)
18     {
19         parent::__construct($registry, WeatherData::class);
20     }
21
22     1 usage new *
23     public function findByLocation(Location $location)
24     {
25         $qb = $this->createQueryBuilder(alias: 'wd'); // używamy aliasu 'wd' dla WeatherData
26         $qb->where(...predicates: 'wd.location = :location')
27             ->setParameter(key: 'location', $location)
28             ->andWhere(...where: 'wd.date > :now') // sprawdzamy, czy data jest większa niż dzisiaj
29             ->setParameter(key: 'now', date(format: 'Y-m-d'));
30
31         $query = $qb->getQuery();
32         $result = $query->getResult();
33         return $result;
34     }
35 }

```

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/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.