

API I TESTY JEDNOSTKOWE

SPIS TREŚCI

Spis treści	1
Cel zajęć.....	1
Rozpoczęcie	1
Uwaga	1
Serwie WeatherUtil.....	Błąd! Nie zdefiniowano zakładki.
Komendy	Błąd! Nie zdefiniowano zakładki.
Commit projektu do GIT.....	10
Podsumowanie.....	16

CEL ZAJĘĆ

Celem głównym zajęć jest zdobycie następujących umiejętności:

- wykorzystanie reużywalnej logiki biznesowej z serwisów do tworzenia API;
- testowanie jednostkowe.

ROZPOCZĘCIE

Rozpoczęcie zajęć. Powtórzenie możliwości tworzenia API w Symfony. Omówienie testów jednostkowych, integracyjnych i funkcjonalnych.

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**.

ODCZYT WEJŚCIA W KONTROLERZE

Wykorzystaj komendę `make:controller` do stworzenia kontrolera `WeatherApiController`:

```
php .\bin\console make:controller --no-template
```

```
Choose a name for your controller class (e.g. AgreeablePuppyController):
> WeatherApiController

created: src/Controller/WeatherApiController.php

Success!

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

Zastosowanie flagi `--no-template` skutkuje wygenerowaniem kontrolera, który zwraca JSON zamiast renderowania szablonu:

```
class WeatherApiController extends AbstractController
{
    #[Route('/weather/api', name: 'app_weather_api')]
    public function index(): JsonResponse
    {
        return $this->json([
            'message' => 'Welcome to your new controller!',
            'path' => 'src/Controller/WeatherApiController.php',
        ]);
    }
}
```

Zmień ścieżkę routingu na „`/api/v1/weather`”, metoda pozostaje GET.

Wykorzystaj atrybuty `MapQueryParameter` do zmapowania parametrów `country` i `city` do ustawienia lokalnych zmiennych `$country` i `$city`. Więcej informacji: <https://symfony.com/blog/new-in-symfony-6-3-query-parameters-mapper>.

Na ten moment działanie kontrolera ogranicz do wyświetlenia w JSONie otrzymanych parametrów wejściowych:

```
return $this->json([
    'city' => $city,
    'country' => $country,
]);
```

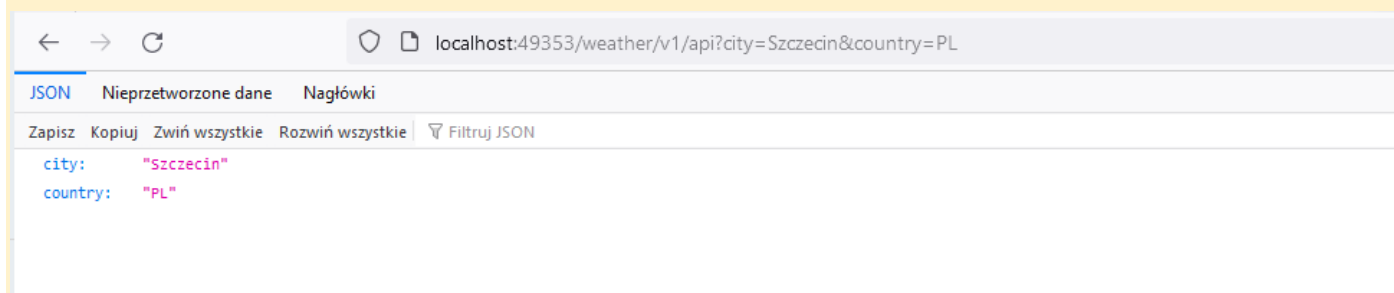
Wstaw zrzut ekranu kodu kontrolera na tym etapie:

```

ogodynka > src > Controller > WeatherApiController.php > PHP Intelephense > WeatherApiController > index
1  <?php
2
3  namespace App\Controller;
4
5  use Symfony\Component\Routing\Annotation\Route;
6  use Symfony\Component\HttpFoundation\JsonResponse;
7  use Symfony\Component\HttpFoundation\Request;
8  use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
9
10 class WeatherApiController extends AbstractController
11 {
12     #[Route('/weather/v1/api', name: 'app_weather_api', methods: ['GET'])]
13     public function index(
14         #[MapQueryParameter] string $city,
15         #[MapQueryParameter] string $country
16     ): JsonResponse {
17         return $this->json([
18             'city' => $city,
19             'country' => $country,
20         ]);
21     }
22 }
23

```

Wstaw zrzut ekranu otrzymanego z kontrolera JSONa:



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

WYKORZYSTANIE SERWISU

Podłącz do akcji kontrolera serwis WeatherUtil. Wykorzystaj go do pobrania prognozy pogody dla zadanej miejscowości, a następnie uzupełnij JSON wynikowy o pozycję 'measurements' – tablicę wyników.

Przydatny może się okazać kod z wykorzystaniem array_map:

```
'measurements' => array_map(fn(Measurement $m) => [
```

```

        'date' => $m->getDate()->format('Y-m-d'),
        'celsius' => $m->getCelsius(),
    ], $measurements),

```

Wstaw zrzut ekranu kodu kontrolera na tym etapie:

```

pogodynka > src > Controller > WeatherApiController.php > PHP Intelephense > WeatherApiController > index
1  <?php
2
3  namespace App\Controller;
4
5  use App\Service\WeatherUtil;
6  use Symfony\Component\Routing\Annotation\Route;
7  use Symfony\Component\HttpFoundation\JsonResponse;
8  use Symfony\Component\HttpKernel\Attribute\MapQueryParameter;
9  use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
10
11 class WeatherApiController extends AbstractController
12 {
13     #[Route('/weather/v1/api', name: 'app_weather_api', methods: ['GET'])]
14     public function index(
15         WeatherUtil $util,
16         #[MapQueryParameter] string $city,
17         #[MapQueryParameter] string $country
18     ): JsonResponse {
19         $measurements = $util->getWeatherForCountryAndCity($country, $city);
20         // only dates and temperatures
21         $measurements = array_map(function ($measurement) {
22             return [
23                 'date' => $measurement->getDate()->format('Y-m-d'),
24                 'temperature' => $measurement->getCelsius(),
25             ];
26         }, $measurements);
27
28         return $this->json([
29             'city' => $city,
30             'country' => $country,
31             'measurements' => $measurements,
32         ]);
33     }
34 }
35
36

```

Wstaw zrzuty ekranu otrzymanego z kontrolera JSONa dla dwóch miejscowości:

localhost:49353/weather/v1/api?city=Szczecin&country=PL

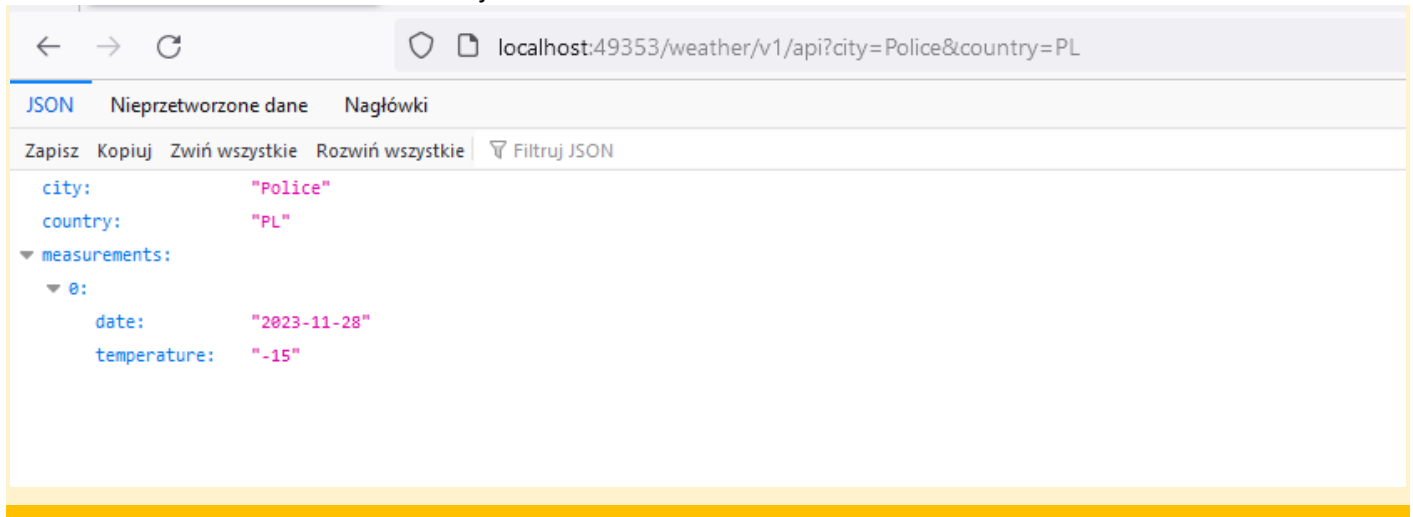
JSON Nieprzetworzone dane Nagłówki

Zapisz Kopiuj Zwiń wszystkie Rozwiń wszystkie Filtruj JSON

```

city: "Szczecin"
country: "PL"
measurements:
  0:
    date: "2023-11-28"
    temperature: "-3"

```



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

FORMAT CSV

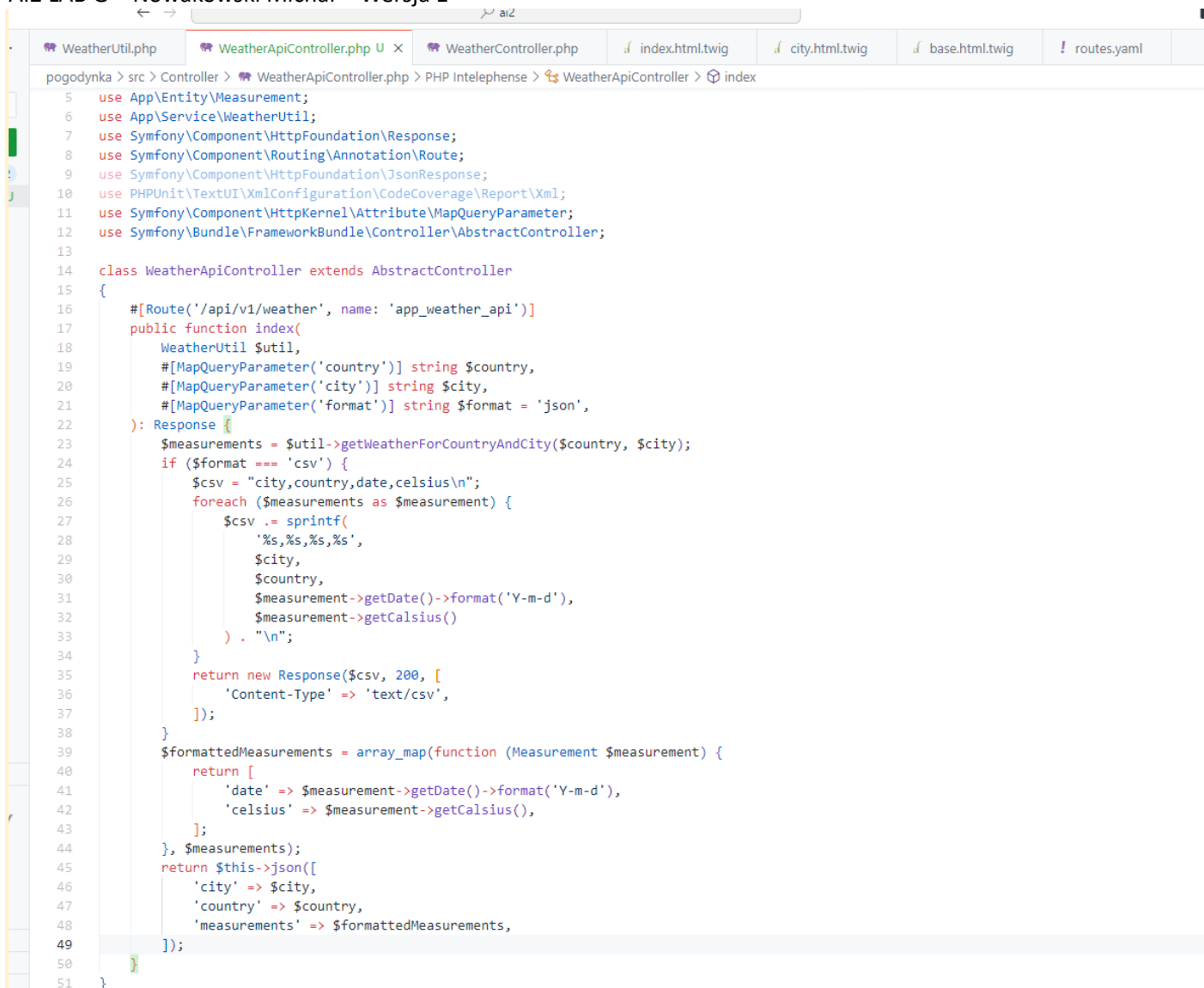
Uzupełnij przyjmowane przez akcję kontrolera parametry o parametr format. Dopuszczalne wartości to json i csv. Dla json działanie kontrolera zostaje jak poprzednio. Dla csv zwrócony powinien zostać wynik w postaci rozdzielanej przecinkami, o kolumnach:

- city
- country
- date
- celsius

Zwróć uwagę, że city i country podawane będą redundantnie w każdej linii.

Wykorzystaj funkcję sprintf() albo implode.

Wstaw zrzut ekranu kodu kontrolera na tym etapie:

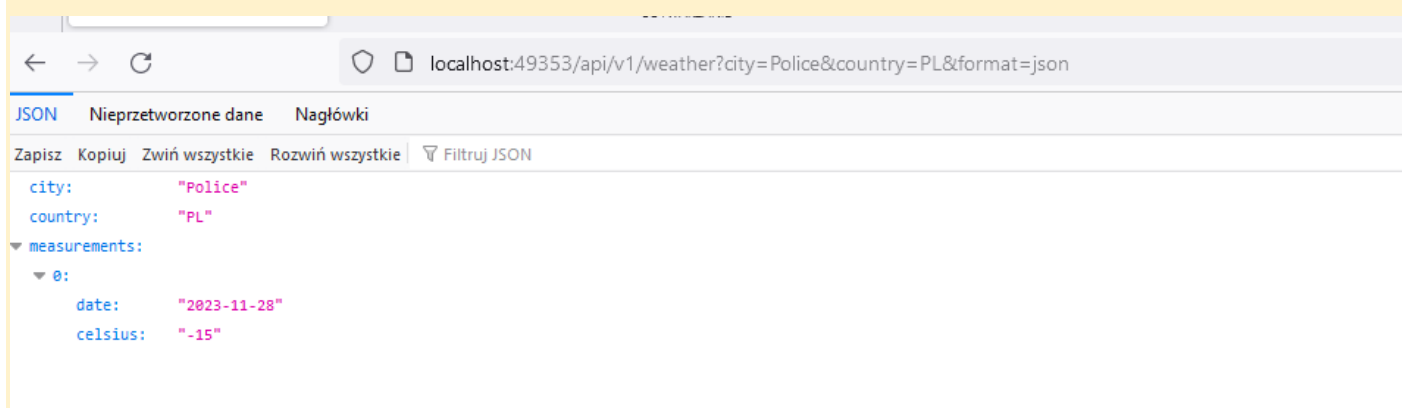


```

5 use App\Entity\Measurement;
6 use App\Service\WeatherUtil;
7 use Symfony\Component\HttpFoundation\Response;
8 use Symfony\Component\Routing\Annotation\Route;
9 use Symfony\Component\HttpFoundation\JsonResponse;
10 use PHPUnit\TextUI\XmlConfiguration\CodeCoverage\Report\Xml;
11 use Symfony\Component\HttpKernel\Attribute\MapQueryParameter;
12 use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
13
14 class WeatherApiController extends AbstractController
15 {
16     #[Route('/api/v1/weather', name: 'app_weather_api')]
17     public function index(
18         WeatherUtil $util,
19         #[MapQueryParameter('country')] string $country,
20         #[MapQueryParameter('city')] string $city,
21         #[MapQueryParameter('format')] string $format = 'json',
22     ): Response {
23         $measurements = $util->getWeatherForCountryAndCity($country, $city);
24         if ($format === 'csv') {
25             $csv = "city,country,date,celsius\n";
26             foreach ($measurements as $measurement) {
27                 $csv .= sprintf(
28                     '%s,%s,%s,%s',
29                     $city,
30                     $country,
31                     $measurement->getDate()->format('Y-m-d'),
32                     $measurement->getCelsius()
33                 ) . "\n";
34             }
35             return new Response($csv, 200, [
36                 'Content-Type' => 'text/csv',
37             ]);
38         }
39         $formattedMeasurements = array_map(function (Measurement $measurement) {
40             return [
41                 'date' => $measurement->getDate()->format('Y-m-d'),
42                 'celsius' => $measurement->getCelsius(),
43             ];
44         }, $measurements);
45         return $this->json([
46             'city' => $city,
47             'country' => $country,
48             'measurements' => $formattedMeasurements,
49         ]);
50     }
51 }

```

Wstaw zrzut ekranu otrzymanego z kontrolera JSONa:

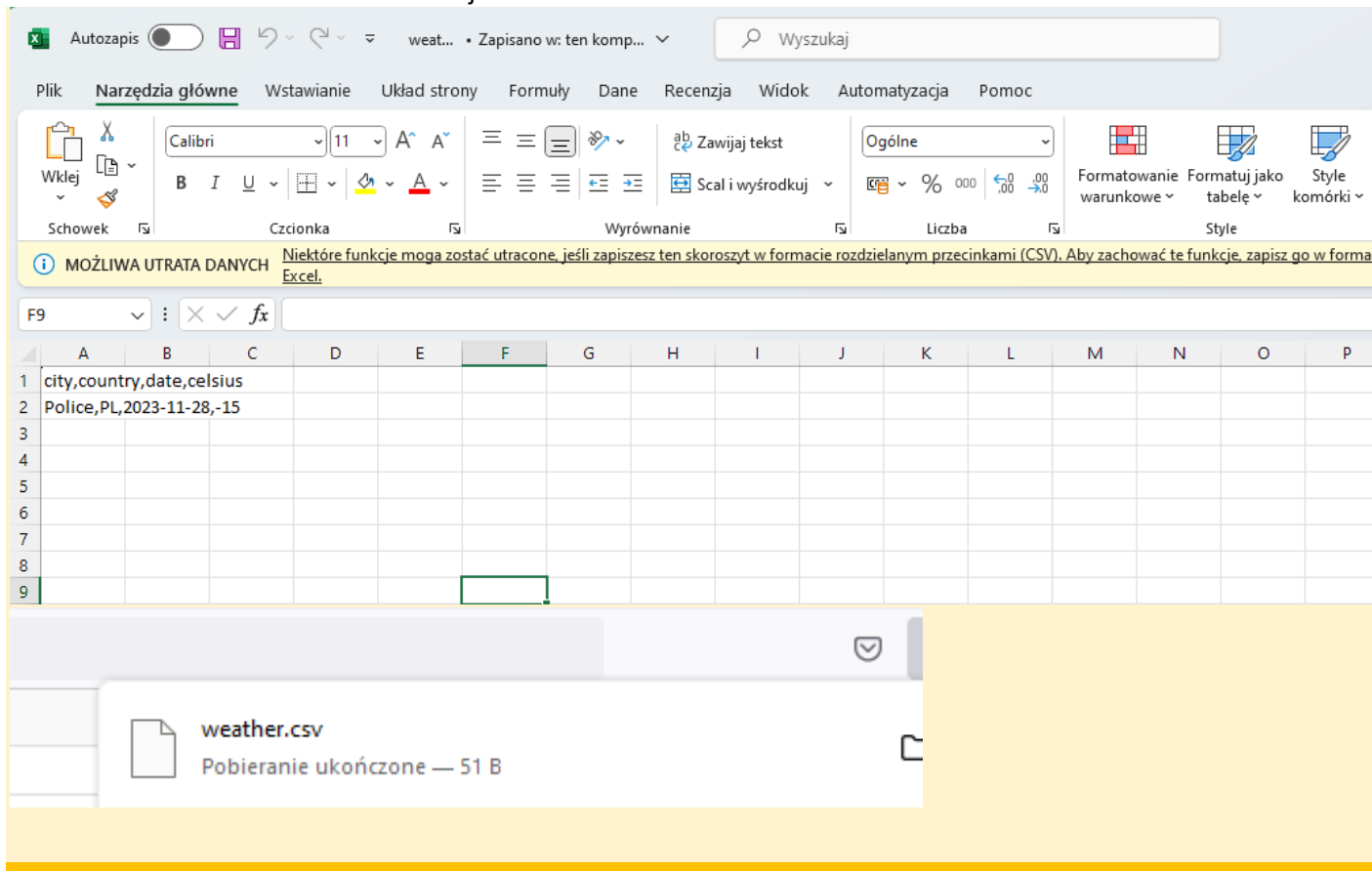


```

{
  "city": "Police",
  "country": "PL",
  "measurements": [
    {
      "date": "2023-11-28",
      "celsius": "-15"
    }
  ]
}

```

Wstaw zrzut ekranu otrzymanego z tego samego kontrolera CSV:



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

WYKORZYSTANIE TWIG

W tej sekcji otrzymamy identyczne wyniki jak w poprzednich sekcjach, z wykorzystaniem szablonów TWIG do generowania odpowiedzi.

Utwórz pliki:

- templates/weather_api/index.csv.twig
- templates/weather_api/index.json.twig

W kontrolerze dodaj nowy opcjonalny parametr boolowski twig. Ustawienie jego wartości skutkować będzie renderowaniem odpowiedzi z wykorzystaniem TWIG:

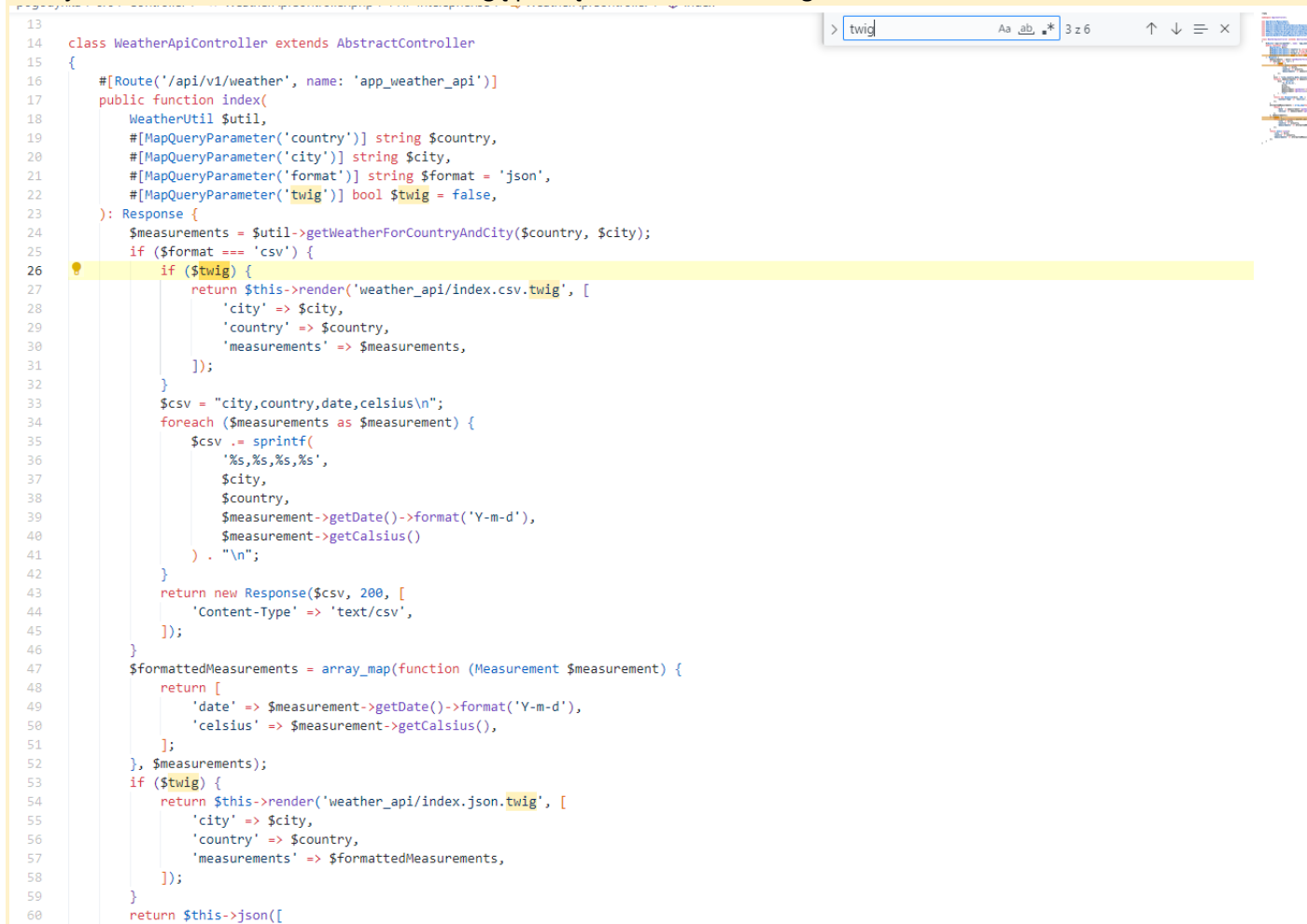
```
#[MapQueryParameter('twig')] bool $twig = false,
```

Przykładowy sposób wywołania generowania odpowiedzi z wykorzystaniem TWIG:

```
return $this->render('weather_api/index.csv.twig', [
    'city' => $city,
    'country' => $country,
    'measurements' => $measurements,
]);
```

Skopiuj teraz odpowiedzi CSV i JSON Twojego API w dotychczasowej wersji i wklej do szablonów TWIG. Następnie, wykorzystaj parametry city, country i measurements oraz instrukcje sterujące TWIG do zamiany tych statycznych odpowiedzi do postaci dynamicznej.

Wklej zrzut ekranu kodu kontrolera z obsługą przełączania formatu i twiga:



```

13
14 class WeatherApiController extends AbstractController
15 {
16     #[Route('/api/v1/weather', name: 'app_weather_api')]
17     public function index(
18         WeatherUtil $util,
19         #[MapQueryParameter('country')] string $country,
20         #[MapQueryParameter('city')] string $city,
21         #[MapQueryParameter('format')] string $format = 'json',
22         #[MapQueryParameter('twig')] bool $twig = false,
23     ): Response {
24         $measurements = $util->getWeatherForCountryAndCity($country, $city);
25         if ($format === 'csv') {
26             if ($twig) {
27                 return $this->render('weather_api/index.csv.twig', [
28                     'city' => $city,
29                     'country' => $country,
30                     'measurements' => $measurements,
31                 ]);
32             }
33             $csv = "city,country,date,celsius\n";
34             foreach ($measurements as $measurement) {
35                 $csv .= sprintf(
36                     '%s,%s,%s,%s',
37                     $city,
38                     $country,
39                     $measurement->getDate()->format('Y-m-d'),
40                     $measurement->getCelsius()
41                 ) . "\n";
42             }
43             return new Response($csv, 200, [
44                 'Content-Type' => 'text/csv',
45             ]);
46         }
47         $formattedMeasurements = array_map(function (Measurement $measurement) {
48             return [
49                 'date' => $measurement->getDate()->format('Y-m-d'),
50                 'celsius' => $measurement->getCelsius(),
51             ];
52         }, $measurements);
53         if ($twig) {
54             return $this->render('weather_api/index.json.twig', [
55                 'city' => $city,
56                 'country' => $country,
57                 'measurements' => $formattedMeasurements,
58             ]);
59         }
60         return $this->json([

```

Punkty:

0

1

Wklej zrzut ekranu kodu TWIG generowania odpowiedzi w formacie JSON:


```

40     }
47     $formattedMeasurements = array_map(function (Measurement $measurement) {
48         return [
49             'date' => $measurement->getDate()->format('Y-m-d'),
50             'celsius' => $measurement->getCelsius(),
51         ];
52     }, $measurements);
53     if ($twig) {
54         return $this->render('weather_api/index.json.twig', [
55             'city' => $city,
56             'country' => $country,
57             'measurements' => $formattedMeasurements,
58         ]);
59     }
60     return $this->json([
61         'city' => $city,
62         'country' => $country,
63         'measurements' => $formattedMeasurements,
64     ]);
65 }
66 }
67

```

Wklej zrzut ekranu przykładowej odpowiedzi JSON wygenerowanej przez TWIG. Upewnij się, że zrzut ekranu zawiera całość adresu URL ze wszystkimi parametrami wywołania (&format=json&twig=1).



localhost:49353/api/v1/weather?city=Police&country=PL&format=json&twig=true

```
{ "city": "Police", "country": "PL", "measurements": [ { "date": "2023-11-28", "celsius": "-15" } ] }
```

Punkty:

0

1

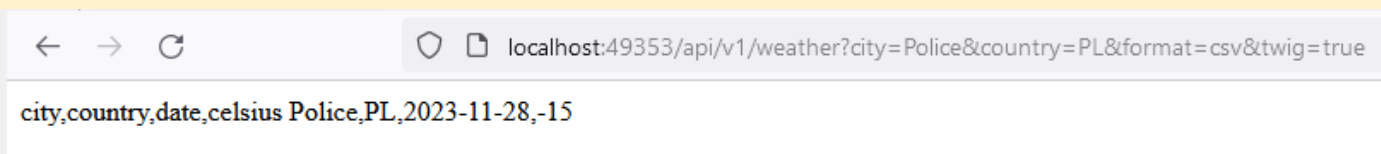
Wklej zrzut ekranu kodu TWIG generowania odpowiedzi w formacie CSV:

```

14 class WeatherApiController extends AbstractController
15 {
16     #[Route('/api/v1/weather', name: 'app_weather_api')]
17     public function index(
18         WeatherUtil $util,
19         #[MapQueryParameter('country')] string $country,
20         #[MapQueryParameter('city')] string $city,
21         #[MapQueryParameter('format')] string $format = 'json',
22         #[MapQueryParameter('twig')] bool $twig = false,
23     ): Response {
24         $measurements = $util->getWeatherForCountryAndCity($country, $city);
25         if ($format === 'csv') {
26             if ($twig) {
27                 return $this->render('weather_api/index.csv.twig', [
28                     'city' => $city,
29                     'country' => $country,
30                     'measurements' => $measurements,
31                 ]);
32             }
33             $csv = "city,country,date,celsius\n";
34             foreach ($measurements as $measurement) {
35                 $csv .= sprintf(
36                     '%s,%s,%s,%s',
37                     $city,
38                     $country,
39                     $measurement->getDate()->format('Y-m-d'),
40                     $measurement->getCelsius()
41                 ) . "\n";
42             }
43             return new Response($csv, 200, [
44                 'Content-Type' => 'text/csv',
45             ]);
46         }

```

Wklej zrzut ekranu przykładowej odpowiedzi CSV wygenerowanej przez TWIG. Upewnij się, że zrzut ekranu zawiera całość adresu URL ze wszystkimi parametrami wywołania (&format=json&twig=1).



Punkty:

0

1

FAHRENHEIT

Do encji pomiarów dodaj metodę `getFahrenheit()`. Metoda ta powinna zwracać wartość `$this->getCelsius()` skonwertowaną do skali Fahrenheita. Formuła: $(0^{\circ}\text{C} \times 9/5) + 32 = 32^{\circ}\text{F}$

Zmodyfikuj wszystkie cztery odpowiedzi API (JSON, CSV), aby zwracały temperaturę w skali Celsjusza i Fahrenheita, przykładowo:

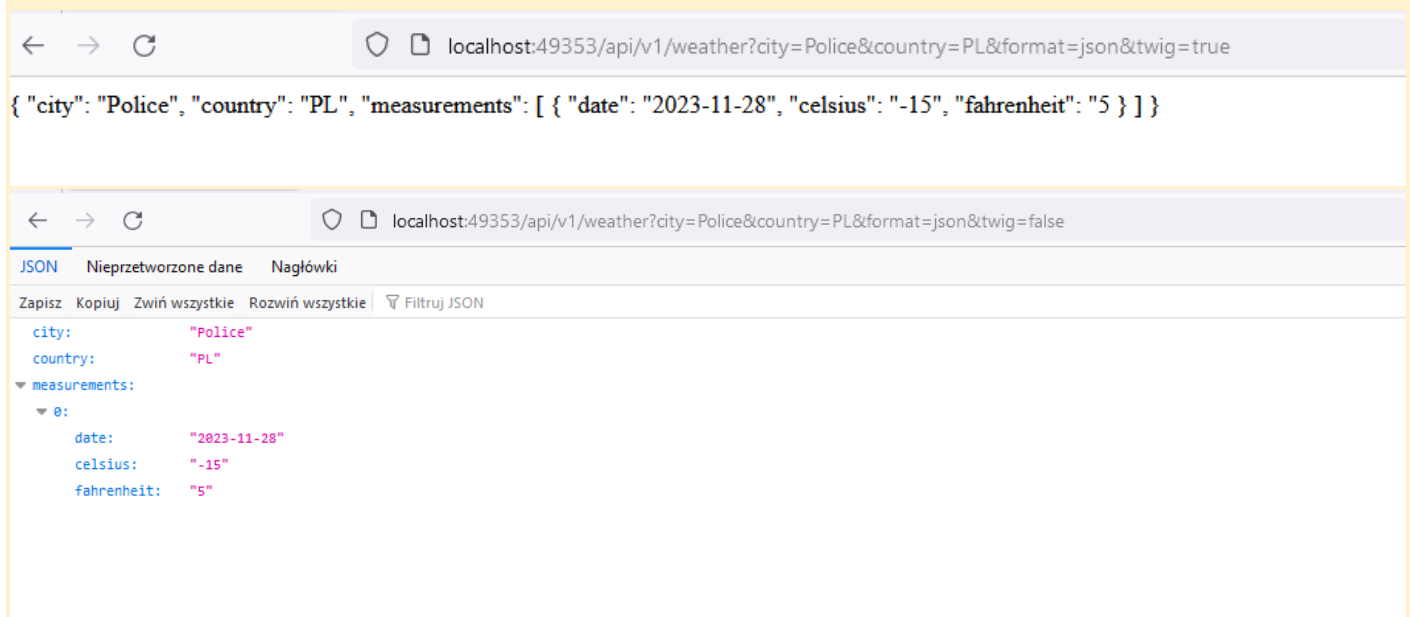
```
//...
'date' => $m->getDate()->format('Y-m-d'),
'celsius' => $m->getCelsius(),
'fahrenheit' => $m->getFahrenheit(),
//...
```

Wklej zrzut ekranu kodu metody getFahrenheit():

```
67
68
69
70
71
72
73

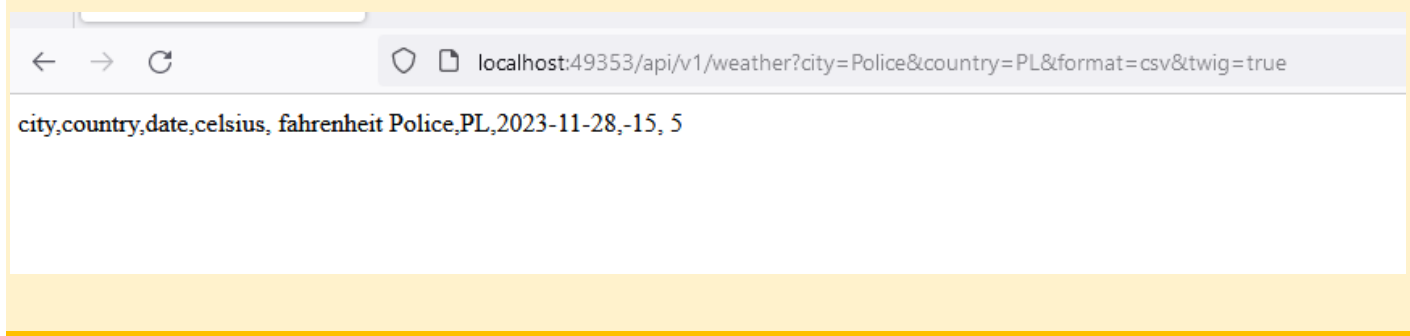
public function getFahrenheit(): ?string
{
    return $this->celsius === null ? null : (string)round($this->celsius * 1.8 + 32);
}
```

Wklej zrzut ekranu przykładowej odpowiedzi JSON z uwzględnieniem obu skali temperatury:



The screenshot shows a web browser at the URL `localhost:49353/api/v1/weather?city=Police&country=PL&format=json&twig=true`. The response is a JSON object: `{ "city": "Police", "country": "PL", "measurements": [{ "date": "2023-11-28", "celsius": "-15", "fahrenheit": "5" }] }`. Below this, there is another screenshot of the same API endpoint but with `format=json&twig=false`. This view shows a structured JSON representation with expandable sections for 'city', 'country', and 'measurements'. The 'measurements' section is expanded, showing an array with one object containing 'date', 'celsius', and 'fahrenheit' values.

Wklej zrzut ekranu przykładowej odpowiedzi CSV z uwzględnieniem obu skali temperatury:



The screenshot shows a web browser at the URL `localhost:49353/api/v1/weather?city=Police&country=PL&format=csv&twig=true`. The response is a CSV string: `city,country,date,celsius, fahrenheit Police,PL,2023-11-28,-15, 5`. The first part of the string represents the headers, and the second part represents the data row.

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

TEST JEDNOSTKOWY

Wykorzystaj metodę `make:test` do utworzenia szablonu testu jednostkowego:

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

Which test type would you like?:
[TestCase      ] basic PHPUnit tests
[KernelTestCase] basic tests that have access to Symfony services
[WebTestCase   ] to run browser-like scenarios, but that don't execute JavaScript code
[ApiTestCase   ] to run API-oriented scenarios
[PantherTestCase] to run e2e scenarios, using a real-browser or HTTP client and a real web server
> TestCase

Choose a class name for your test, like:
* UtilTest (to create tests/UtilTest.php)
* Service\UtilTest (to create tests/Service/UtilTest.php)
* \App\Tests\Service\UtilTest (to create tests/Service/UtilTest.php)

The name of the test class (e.g. BlogPostTest):
> Entity\MeasurementTest

created: tests/Entity/MeasurementTest.php

Success!

Next: Open your new test class and start customizing it.
Find the documentation at https://symfony.com/doc/current/testing.html#unit-tests
```

Zmień metodę `testSomething()` w utworzonym `tests/Entity/MeasurementTest.php` na `testGetFahrenheit`.

Zaimplementuj test, który utworzy nową encję pomiarów, a następnie kolejno:

- ustawi wartość stopni Celsjusza na 0 i sprawdzi czy `getFahrenheit()` zwraca poprawną wartość;
- ustawi wartość stopni Celsjusza na -100 i sprawdzi czy `getFahrenheit()` zwraca poprawną wartość;
- ustawi wartość stopni Celsjusza na 100 i sprawdzi czy `getFahrenheit()` zwraca poprawną wartość.

Pamiętaj, aby wartości stopni Celsjusza i Fahrenheita wewnątrz testu przekazywać jako wartości wpisane „na sztywno”, a nie wyliczane.

Pamiętaj, aby wartości stopni Celsjusza i Fahrenheita wewnątrz testu przekazywać jako wartości wpisane „na sztywno”, a nie wyliczane.

...maksymalnie dwa zdania wytłumaczenia dlaczego nie można użyć `getFahrenheit()` do wyliczenia wartości „w locie”...

Dlatego że to złe podejście, w ten sposób nie musimy się przejmować bazą danych, czy istnieje czy ma dobre wartości itp.

Uruchom test z wykorzystaniem komendy:

```
php .\bin\phpunit
PHPUnit 9.6.13 by Sebastian Bergmann and contributors.

Testing
.
1 / 1 (100%)

Time: 00:00.230, Memory: 6.00 MB

OK (1 test, 3 assertions)
```

Wklej zrzuty ekranu całości kodu pliku MeasurementTest.php:

```
MeasurementTest.php U X
pogodynka > tests > Entity > MeasurementTest.php > ...
1  <?php
2  |
3  namespace App\Tests\Entity;
4
5  use App\Entity\Measurement;
6  use App\Entity\Location;
7  use PHPUnit\Framework\TestCase;
8
9  class MeasurementTest extends TestCase
10 {
11     public function testGetFahrenheit(): void
12     {
13         $measurement = new Measurement();
14         $measurement->setCelsius('0');
15         $this->assertEquals('32', $measurement->getFahrenheit());
16
17         $measurement->setCelsius('-100');
18         $this->assertEquals('-148', $measurement->getFahrenheit());
19
20         $measurement->setCelsius('100');
21         $this->assertEquals('212', $measurement->getFahrenheit());
22     }
23 }
24
```

Wklej zrzut ekranu wywołania i wyniku testów:

```

OK (1 test, 1 assertion)
PS C:\Users\micha\Documents\GitHub\ai2\pogodynka> php .\bin\phpunit
PHPUnit 9.6.13 by Sebastian Bergmann and contributors.

Testing
.
1 / 1 (100%)

Time: 00:00.054, Memory: 6.00 MB

OK (1 test, 3 assertions)
PS C:\Users\micha\Documents\GitHub\ai2\pogodynka> 

```

Upewnij się że wykonano 1 test i 3 asercje.

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

DATAPROVIDER

W tej sekcji sprawdzimy więcej przypadków, również uwzględniających ułamki. Wykorzystamy dataProvider. Utwórz funkcję dataGetFahrenheit():

```

public function dataGetFahrenheit(): array
{
    return [
        ['0', 32],
        ['-100', -148],
        ['100', 212],
    ];
}

```

Nad testem dodaj adnotację:

```
@dataProvider dataGetFahrenheit
```

Zmień sygnaturę funkcji testu:

```
public function testGetFahrenheit($celsius, $expectedFahrenheit): void
```

Zmodyfikuj kod funkcji w taki sposób, żeby zamiast „na sztywno” sprawdzać wartości 0, -100 i 100, wykorzystywał parametr \$celsius i \$expectedFahrenheit.

Uzupełnij dane wejściowe w dataGetFahrenheit do 10 wartości, również wykorzystujących ułamki, np. 0.5 stopnia Celsjusza to 32.9 stopnia Fahrenheita.

Wklej zrzuty ekranu całości kodu pliku MeasurementTest.php:

```

1  <?php
2
3  namespace App\Tests\Entity;
4
5  use App\Entity\Measurement;
6  use App\Entity\Location;
7  use PHPUnit\Framework\TestCase;
8
9  class MeasurementTest extends TestCase
10 {
11     public function dataGetFarhenheit(): array
12     {
13         return [
14             ['0', 32],
15             ['10', 50],
16             ['15', 59],
17             ['20', 68],
18             ['25', 77],
19             ['30', 86],
20             ['35', 95],
21             ['40', 104],
22             ['45', 113],
23             ['50', 122],
24             ['55', 131],
25             ['60', 140],
26             ['65', 149],
27             ['70', 158],
28             ['75', 167],
29             ['80', 176],
30             ['85', 185],
31             ['90', 194],
32             ['95', 203],
33             ['100', 212],
34         ];
35     }
36
37     /**
38      * @dataProvider dataGetFarhenheit
39      */
40     public function testGetFahrenheit($celsius, $expectedFahrenheit): void
41     {
42         $measurement = new Measurement();
43         $measurement->setCalsius($celsius);
44         $this->assertEquals($expectedFahrenheit, $measurement->getFahrenheit(), "Expected $expectedFahrenheit Fahrenheit for $celsius Celsius, got {$measurement->getFahrenheit()}");
45     }
46 }
47

```

Wklej zrzut ekranu wywołania i wyniku testów:

```

OK (20 tests, 20 assertions)
● PS C:\Users\micha\Documents\GitHub\ai2\pogodynka> php .\bin\phpunit
PHPUnit 9.6.13 by Sebastian Bergmann and contributors.

Testing
.....                               20 / 20 (100%)

Time: 00:00.030, Memory: 6.00 MB

OK (20 tests, 20 assertions)
○ PS C:\Users\micha\Documents\GitHub\ai2\pogodynka>

```

Upewnij się że wykonano 10 testów i 10 asercji.

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-g na podstawie głównej gałęzi kodu.

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

<https://github.com/Michaldariusznowakowski/ai2/tree/lab-g>

...link, np. <https://github.com/ideaspot-pl/ai2-pogodynka-202310/tree/lab-g...>

PODSUMOWANIE

W kilku zdaniach podsumuj zdobyte podczas tego laboratorium umiejętności.

Podczas tych laboratoriów nauczyłem się jak stworzyć własne api, oraz jak zwracać różne formaty.

Dodatkowo nauczyłem się tworzyć podstawowe testy.

Zweryfikuj kompletność sprawozdania. Utwórz PDF i wyślij w terminie.