

API I TESTY JEDNOSTKOWE

SPIS TREŚCI

| | |
|-----------------------------------|----|
| Spis treści | 1 |
| Cel zajęć..... | 1 |
| Rozpoczęcie | 1 |
| Uwaga | 1 |
| Odczyt wejścia w kontrolerze..... | 2 |
| Wykorzystanie serwisu..... | 3 |
| Format CSV | 5 |
| Wykorzystanie TWIG | 8 |
| Fahrenheit | 11 |
| Test jednostkowy..... | 12 |
| DataProvider | 14 |
| Commit projektu do GIT..... | 16 |
| 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 --no-template` 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:

```

1  <?php
2
3  namespace App\Controller;
4
5  use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
6  use Symfony\Component\HttpFoundation\JsonResponse;
7  use Symfony\Component\Routing\Attribute\Route;
8  use Symfony\Component\HttpKernel\Attribute\MapQueryString;
9
10 final class WeatherApiController extends AbstractController
11 {
12     #[Route('/api/v1/weather', name: 'app_weather_api', methods: ['GET'])]
13     public function getWeather(
14         #[MapQueryString] string $country,
15         #[MapQueryString] string $city,
16     ): JsonResponse
17     {
18         return $this->json([
19             'city' => $city,
20             'country' => $country,
21         ]);
22     }
23 }
24

```

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),

```

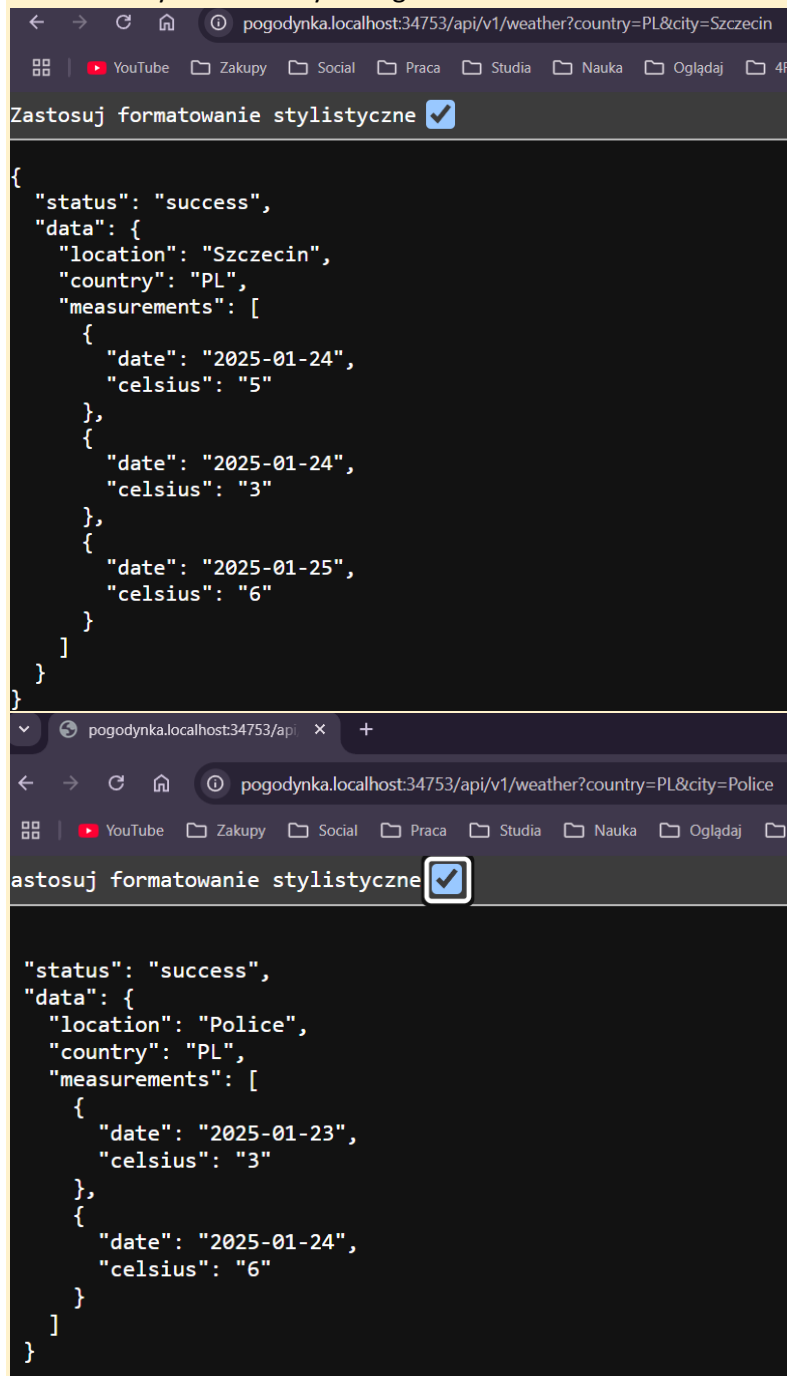
Wstaw zrzut ekranu kodu kontrolera na tym etapie:

```

12 final class WeatherApiController extends AbstractController
13 {
14     2 usages
15     private WeatherUtil $weatherUtil;
16     2 usages
17     private LocationRepository $locationRepository;
18
19     public function __construct(WeatherUtil $weatherUtil, LocationRepository $locationRepository)
20     {
21         $this->weatherUtil = $weatherUtil;
22         $this->locationRepository = $locationRepository;
23     }
24
25     #[Route('/api/v1/weather', name: 'app_weather_api', methods: ['GET'])]
26     public function getWeather(
27         #[MapQueryParameter] string $country,
28         #[MapQueryParameter] string $city,
29     ): JsonResponse
30     {
31         // Pobranie lokalizacji z bazy danych na podstawie kraju i miasta
32         $location = $this->locationRepository->findOneBy([
33             'country' => $country,
34             'city' => $city,
35         ]);
36
37         if (!$location) {
38             return new JsonResponse([
39                 'status' => 'error',
40                 'message' => 'Location not found.',
41             ], status: 404);
42         }
43
44         // Pobranie prognozy pogody za pomoca serwisu WeatherUtil
45         $measurements = $this->weatherUtil->getWeatherForLocation($location);
46
47         // Przekształcenie wyników prognozy na wymagany format
48         $measurementsData = array_map(fn($m) => [
49             'date' => $m->getDate()->format('Y-m-d'),
50             'celsius' => $m->getCelsius(),
51         ], $measurements);
52
53         // Przygotowanie odpowiedzi w formacie JSON
54         return new JsonResponse([
55             'status' => 'success',
56             'data' => [
57                 'location' => $city,
58                 'country' => $country,
59                 'measurements' => $measurementsData,
60             ],
61         ], status: 200);
62     }
63 }

```

Wstaw zrzuty ekranu otrzymanego z kontrolera JSONa dla dwóch miejscowości:



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:

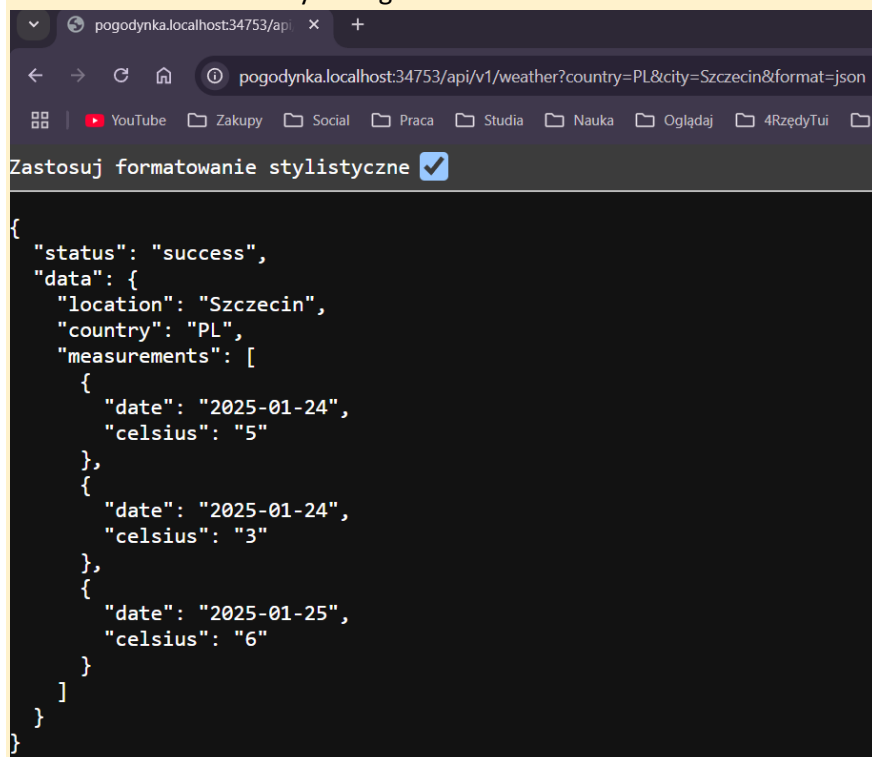
```
13 final class WeatherApiController extends AbstractController
14 {
15     2 usages
16     private WeatherUtil $weatherUtil;
17     2 usages
18     private LocationRepository $locationRepository;
19
20     public function __construct(WeatherUtil $weatherUtil, LocationRepository $locationRepository)
21     {
22         $this->weatherUtil = $weatherUtil;
23         $this->locationRepository = $locationRepository;
24     }
25
26     #[Route('/api/v1/weather', name: 'app_weather_api', methods: ['GET'])]
27     public function getWeather(
28         #[MapQueryParameter] string $country,
29         #[MapQueryParameter] string $city,
30         #[MapQueryParameter] string $format = 'json', // Domyślny format to JSON
31     ): Response
32     {
33         // Pobranie lokalizacji z bazy danych na podstawie kraju i miasta
34         $location = $this->locationRepository->findOneBy([
35             'country' => $country,
36             'city' => $city,
37         ]);
38
39         if (!$location) {
40             return new JsonResponse([
41                 'status' => 'error',
42                 'message' => 'Location not found.',
43             ], status: 404);
44         }
45
46         // Pobranie prognozy pogody za pomoca serwisu WeatherUtil
47         $measurements = $this->weatherUtil->getWeatherForLocation($location);
```

```

46 // Przekształcenie wyników prognozy na wymagany format
47 $measurementsData = array_map(fn($m) => [
48     'date' => $m->getDate()->format('Y-m-d'),
49     'celsius' => $m->getCelsius(),
50 ], $measurements);
51
52 // Jeśli format to JSON, zwróć dane w formacie JSON
53 if ($format === 'json') {
54     return new JsonResponse([
55         'status' => 'success',
56         'data' => [
57             'location' => $city,
58             'country' => $country,
59             'measurements' => $measurementsData,
60         ],
61     ]);
62 }
63
64 // Jeśli format to CSV, generuj odpowiedź w formacie CSV
65 if ($format === 'csv') {
66     $csvData = [];
67     // Dodaj nagłówki kolumn
68     $csvData[] = implode(separator: ',', ['city', 'country', 'date', 'celsius']);
69
70     // Dodaj dane pomiarów
71     foreach ($measurementsData as $measurement) {
72         $csvData[] = implode(separator: ',', [
73             $city,
74             $country,
75             $measurement['date'],
76             $measurement['celsius']
77         ]);
78     }
79
80     // Stwórz odpowiedź CSV
81     $csvContent = implode(separator: "\n", $csvData);
82
83     return new Response(
84         $csvContent,
85         status: Response::HTTP_OK,
86         ['Content-Type' => 'text/csv']
87     );
88 }
89
90 // Jeśli format nie jest poprawny, zwróć błąd
91 return new JsonResponse([
92     'status' => 'error',
93     'message' => 'Invalid format. Use "json" or "csv".',
94 ], status: 400);
95 }
96 }

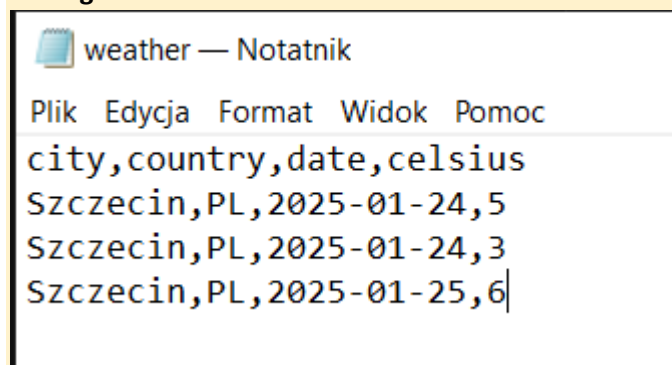
```

Wstaw zrzut ekranu otrzymanego z kontrolera JSONa:



```
{
  "status": "success",
  "data": {
    "location": "Szczecin",
    "country": "PL",
    "measurements": [
      {
        "date": "2025-01-24",
        "celsius": "5"
      },
      {
        "date": "2025-01-24",
        "celsius": "3"
      },
      {
        "date": "2025-01-25",
        "celsius": "6"
      }
    ]
  }
}
```

Wstaw zrzut ekranu otrzymanego z tego samego kontrolera CSV. **Uwaga! Poprawny CSV ma osobny wiersz dla każdego rekordu i NIE ma znaczników
. Jeśli otwierasz wynik w przeglądarce – podejrzuj źródła (Ctrl+U):**



```
weather — Notatnik
Plik Edycja Format Widok Pomoc
city,country,date,celsius
Szczecin,PL,2025-01-24,5
Szczecin,PL,2025-01-24,3
Szczecin,PL,2025-01-25,6
```

| | | |
|---------|---|---|
| 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:

```
#[Route('/api/v1/weather', name: 'app_weather_api', methods: ['GET'])]
public function getWeather(
    #[MapQueryParameter] string $country,
    #[MapQueryParameter] string $city,
    #[MapQueryParameter] string $format = 'json', // Domyślny format to JSON
    #[MapQueryParameter('twig')] bool $twig = false // Parametr twig
): Response
{
    // Jeśli parametr twig jest ustawiony na true, używamy TWIG do renderowania odpowiedzi
    if ($twig) {
        if ($format === 'json') {
            return $this->render( view: 'weather_api/index.json.twig', [
                'city' => $city,
                'country' => $country,
                'measurements' => $measurementsData,
            ]);
        }

        if ($format === 'csv') {
            return $this->render( view: 'weather_api/index.csv.twig', [
                'city' => $city,
                'country' => $country,
                'measurements' => $measurementsData,
            ]);
        }
    }
}
```

Punkty:

0

1

Wklej zrzut ekranu kodu TWIG generowania odpowiedzi w formacie JSON. **UWAGA – renderuj odpowiedź „na piechotę”, a nie używając `json_encode`!**

```

1 {
2     "status": "success",
3     "data": {
4         "location": "{{ city }}",
5         "country": "{{ country }}",
6         "measurements": [
7             {% for measurement in measurements %}
8             {
9                 "date": "{{ measurement.date }}",
10                "celsius": "{{ measurement.celsius }}"
11            }{% if not loop.last %},{% endif %}
12            {% endfor %}
13        ]
14    }
15 }

```

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).

```

{ "status": "success", "data": { "location": "Szczecin", "country": "PL", "measurements": [ { "date": "2025\u002D01\u002D24", "celsius": "5" }, { "date": "2025\u002D01\u002D24", "celsius": "3" }, { "date": "2025\u002D01\u002D25", "celsius": "6" } ] } }

```

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

Wklej zrzut ekranu kodu TWIG generowania odpowiedzi w formacie CSV. **Uwaga! KUDOS za renderowanie jednej linii odpowiedzi CSV w wielu liniach TWIGa – sprawdź funkcjonalność kontroli białych znaków w TWIG:**

```

1 city,country,date,celsius
2 {% for measurement in measurements %}
3 {{ city }},{{ country }},{{ measurement.date }},{{ measurement.celsius }}
4 {% if not loop.last %}\n{% endif %}
5 {% endfor %}

```

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).

```

city,country,date,celsius Szczecin,PL,2025-01-24,5 ,\nSzczecin,PL,2025-01-24,3 ,\nSzczecin,PL,2025-01-25,6

```

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

FAHRENHEIT

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

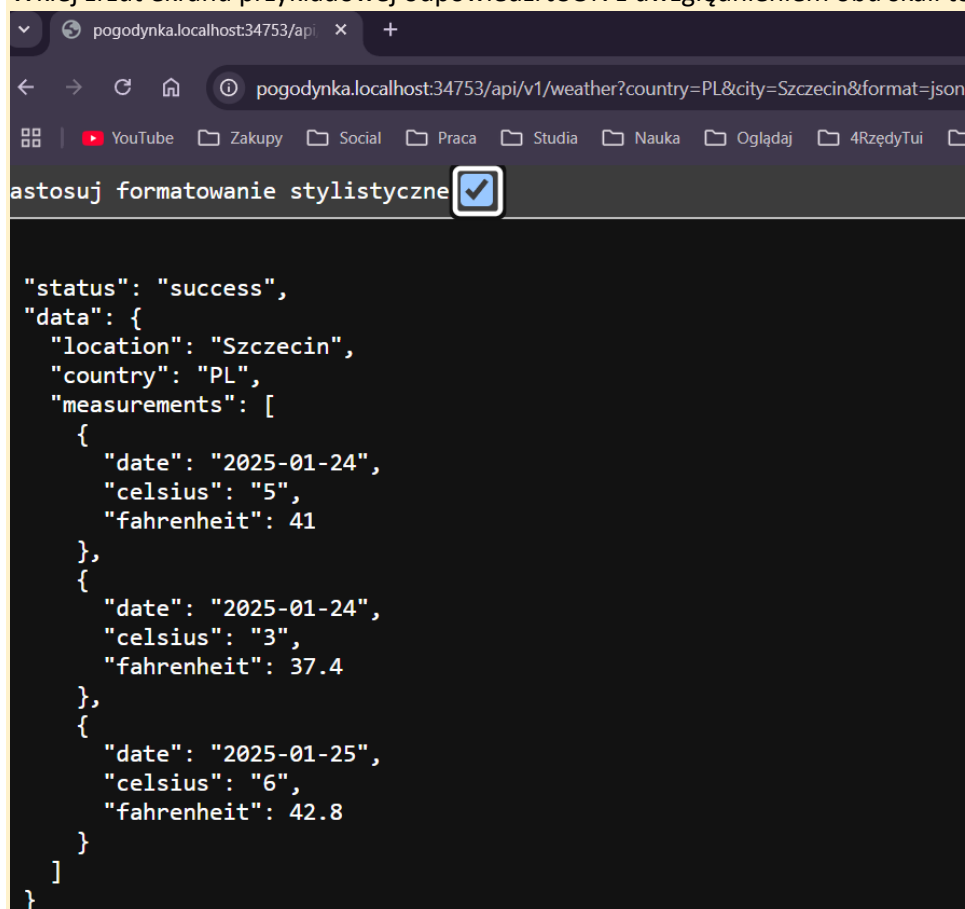
Zmodyfikuj wszystkie cztery odpowiedzi API (JSON, CSV), aby zwracały temperaturę w skali Celsjusza i Fahrenheita, przykładowo:

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

Wklej zrzut ekranu kodu metody `getFahrenheit()`:

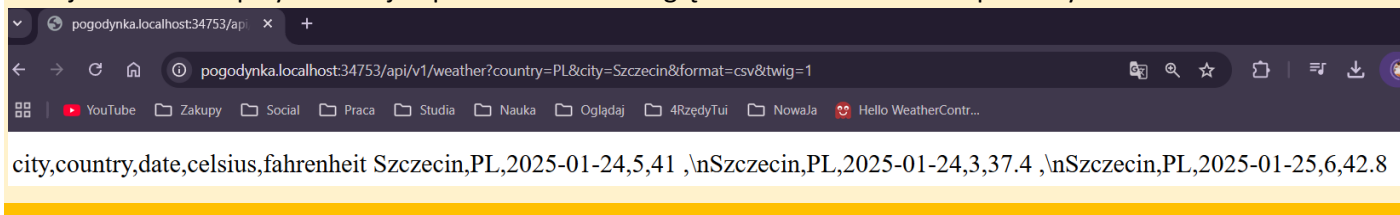
```
74     public function getFahrenheit(): float
75     {
76         return ($this->getCelsius() * 9/5) + 32;
77     }
78
```

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



```
{
  "status": "success",
  "data": {
    "location": "Szczecin",
    "country": "PL",
    "measurements": [
      {
        "date": "2025-01-24",
        "celsius": "5",
        "fahrenheit": 41
      },
      {
        "date": "2025-01-24",
        "celsius": "3",
        "fahrenheit": 37.4
      },
      {
        "date": "2025-01-25",
        "celsius": "6",
        "fahrenheit": 42.8
      }
    ]
  }
}
```

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



| | | |
|---------|---|---|
| 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.

Nie możemy używać `getFahrenheit()` do obliczenia wartości "w locie" w testach, ponieważ metoda ta opiera się na danych już ustawionych w obiekcie (wartości Celsjusza), a testy muszą mieć pełną kontrolę nad wartościami wejściowymi i oczekiwanymi wynikami. Używanie wartości „na sztywno” zapewnia deterministyczność testów i ich niezależność od innych czynników.

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 rzuty ekranu całości kodu pliku `MeasurementTest.php`:

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

Wklej zrzut ekranu wywołania i wyniku testów:

```
PS C:\AI2-lab-v2\pogodynka> php .\bin\phpunit
PHPUnit 9.6.22 by Sebastian Bergmann and contributors.

Testing
F                                                                    1 / 1 (100%)

Time: 00:00.015, Memory: 6.00 MB

There was 1 failure:

1) App\Tests\Entity\MeasurementTest::testGetFahrenheit
Failed asserting that -148.0 matches expected 212.

C:\AI2-lab-v2\pogodynka\tests\Entity\MeasurementTest.php:19
C:\AI2-lab-v2\pogodynka\vendor\phpunit\phpunit\phpunit:107

FAILURES!
Tests: 1, Assertions: 3, Failures: 1.
PS C:\AI2-lab-v2\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 PHPUnit\Framework\TestCase;
7
8  class MeasurementTest extends TestCase
9  {
10     /**
11      * @dataProvider dataGetFahrenheit
12      */
13     public function testGetFahrenheit($celsius, $expectedFahrenheit): void
14     {
15         $measurement = new Measurement();
16         $measurement->setCelsius($celsius); // Używamy wartości z dataProvider
17         $this->assertEquals($expectedFahrenheit, $measurement->getFahrenheit());
18     }
19
20     1 usage
21     public function dataGetFahrenheit(): array
22     {
23         return [
24             // Dodajemy przypadki testowe, również z wartościami ułamkowymi
25             [0, 32],
26             [-100, -148],
27             [100, 212],
28             [0.5, 32.9],
29             [-0.5, 31.1],
30             [25, 77],
31             [-25, -13],
32             [37.5, 99.5],
33             [-50.5, -58.9],
34             [10, 50],
35         ];
36     }
37 }

```

Wklej zrzut ekranu wywołania i wyniku testów:

```

PS C:\AI2-lab-v2\pogodynka> php .\bin\phpunit
PHPUnit 9.6.22 by Sebastian Bergmann and contributors.

Testing
.....E.                                     10 / 10 (100%)

Time: 00:00.014, Memory: 6.00 MB

There was 1 failure:

1) App\Tests\Entity\MeasurementTest::testGetFahrenheit with data set #8 (-50.5, -58.9)
Failed asserting that -58.900000000000006 matches expected -58.9.

C:\AI2-lab-v2\pogodynka\tests\Entity\MeasurementTest.php:17
C:\AI2-lab-v2\pogodynka\vendor\phpunit\phpunit\phpunit:107

FAILURES!
Tests: 10, Assertions: 10, Failures: 1.
PS C:\AI2-lab-v2\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/AleksandraBancewicz/AI2/tree/main/LG>

PODSUMOWANIE

W kilku zdaniach podsumuj zdobyte podczas tego laboratorium umiejętności.

Testowanie konwersji temperatury Celsjusza na Fahrenheita przy użyciu `dataProvider` pozwala na łatwe rozszerzenie zestawu przypadków testowych, co zwiększa pokrycie testami. Wykorzystanie ułamkowych wartości Celsjusza pozwala na dokładniejsze sprawdzenie poprawności obliczeń w różnych scenariuszach. Implementacja takich testów poprawia niezawodność aplikacji, zapewniając, że konwersja temperatur działa prawidłowo w różnych przypadkach. Zastosowanie różnych formatów odpowiedzi (JSON, CSV) oraz szablonów TWIG pozwala na elastyczne generowanie danych w zależności od preferencji użytkownika, co zwiększa uniwersalność aplikacji. Wykorzystanie TWIG do renderowania odpowiedzi umożliwia łatwiejsze dostosowanie formatowania i generowanie treści w bardziej zaawansowany sposób, zapewniając lepszą kontrolę nad wyglądem odpowiedzi.

Zweryfikuj kompletność sprawozdania. Utwórz PDF i wyślij w terminie.