

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	6
Wykorzystanie TWIG	8
Fahrenheit	12
Test jednostkowy	14
DataProvider	16
Commit projektu do GIT.....	18
Podsumowanie	18

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:

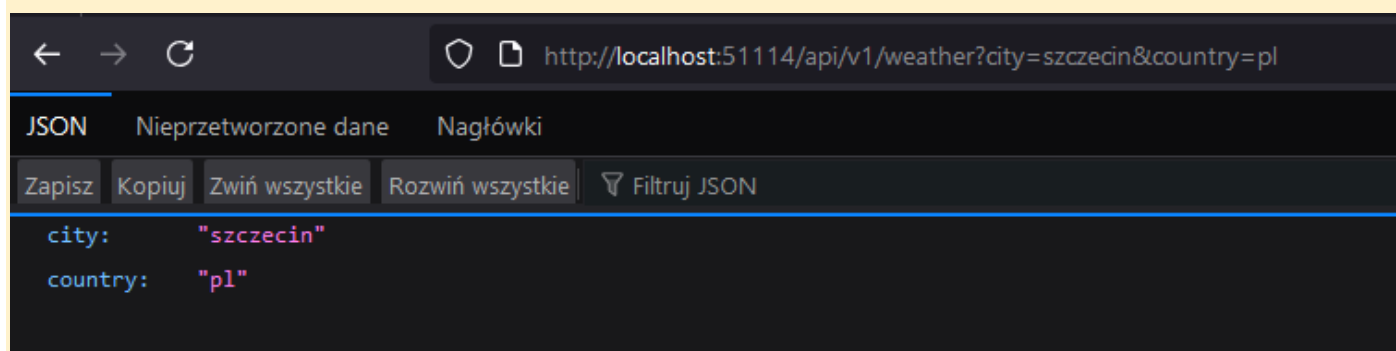
```
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\JsonResponse;
use Symfony\Component\HttpKernel\Attribute\MapQueryParameter;
use Symfony\Component\Routing\Attribute\Route;

You, 12 seconds ago | 1 author (You)
class WeatherApiController extends AbstractController
{
    #[Route('/api/v1/weather', name: 'app_weather_api')]
    public function index(
        #[MapQueryParameter(filter: \FILTER_VALIDATE_REGEXP, options: ['regexp' => '/^[a-zA-Z-]+$/'])] string $city,
        #[MapQueryParameter(filter: \FILTER_VALIDATE_REGEXP, options: ['regexp' => '/^[a-zA-Z]{2}$/'])] string $country,
    ): JsonResponse {
        return $this->json([
            'city' => $city,
            'country' => $country,
        ]);
    }
}
```

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:

```

<?php

namespace App\Controller;

use App\Entity\MeasurementEntry;
use App\Service\WeatherUtil;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\JsonResponse;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\Routing\Annotation\Route;

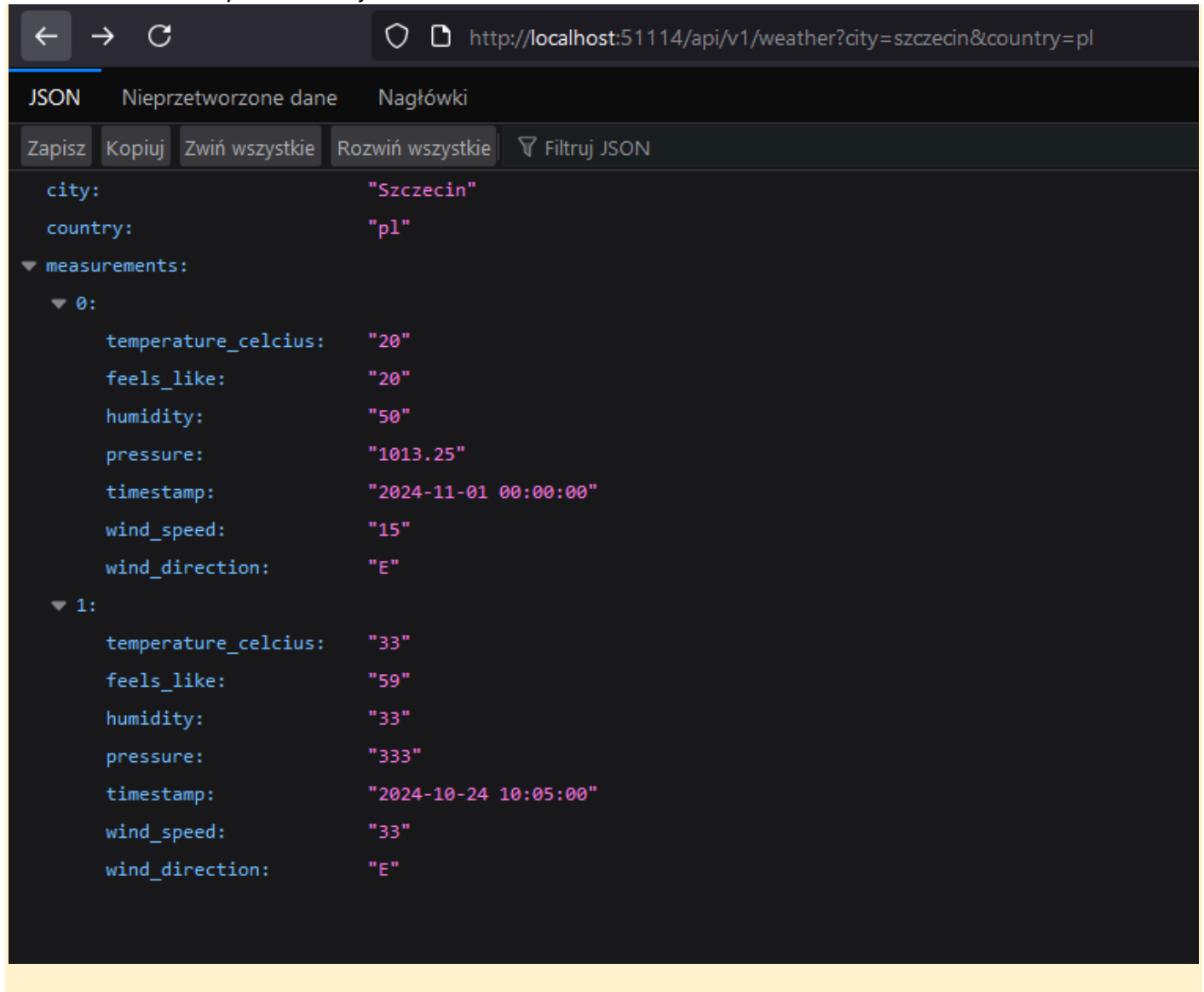
...
class WeatherApiController extends AbstractController
{
    #[Route('/api/v1/weather', name: 'app_weather_api')]
    public function index(
        #[MapQueryStringParameter(filter: \FILTER_VALIDATE_REGEXP, options: ['regexp' => '/^[a-zA-Z-]+$/'])] string $city,
        #[MapQueryStringParameter(filter: \FILTER_VALIDATE_REGEXP, options: ['regexp' => '/^[a-zA-Z]{2}$/'])] string $country,
        WeatherUtil $weatherUtil,
    ): JsonResponse {
        // You, 4 minutes ago * .

        $measurementEntries = $weatherUtil->getWeatherForCountryAndCity($country, $city);

        return $this->json(
            [
                'city' => $city,
                'country' => $country,
                'measurements' => array_map(
                    fn(MeasurementEntry $m) => [
                        'temperature_celcius' => $m->getTemperatureCelcius(),
                        'feels_like' => $m->getFeelsLike(),
                        'humidity' => $m->getHumidity(),
                        'pressure' => $m->getPressure(),
                        'timestamp' => $m->getDateTime()->format('Y-m-d H:i:s'),
                        'wind_speed' => $m->getWindSpeed(),
                        'wind_direction' => $m->getWindDirection(),
                    ],
                    $measurementEntries
                ),
            ],
        );
    }
}

```

Wstaw zrzuty ekranu otrzymanego z kontrolera JSONa dla dwóch miejscowości:

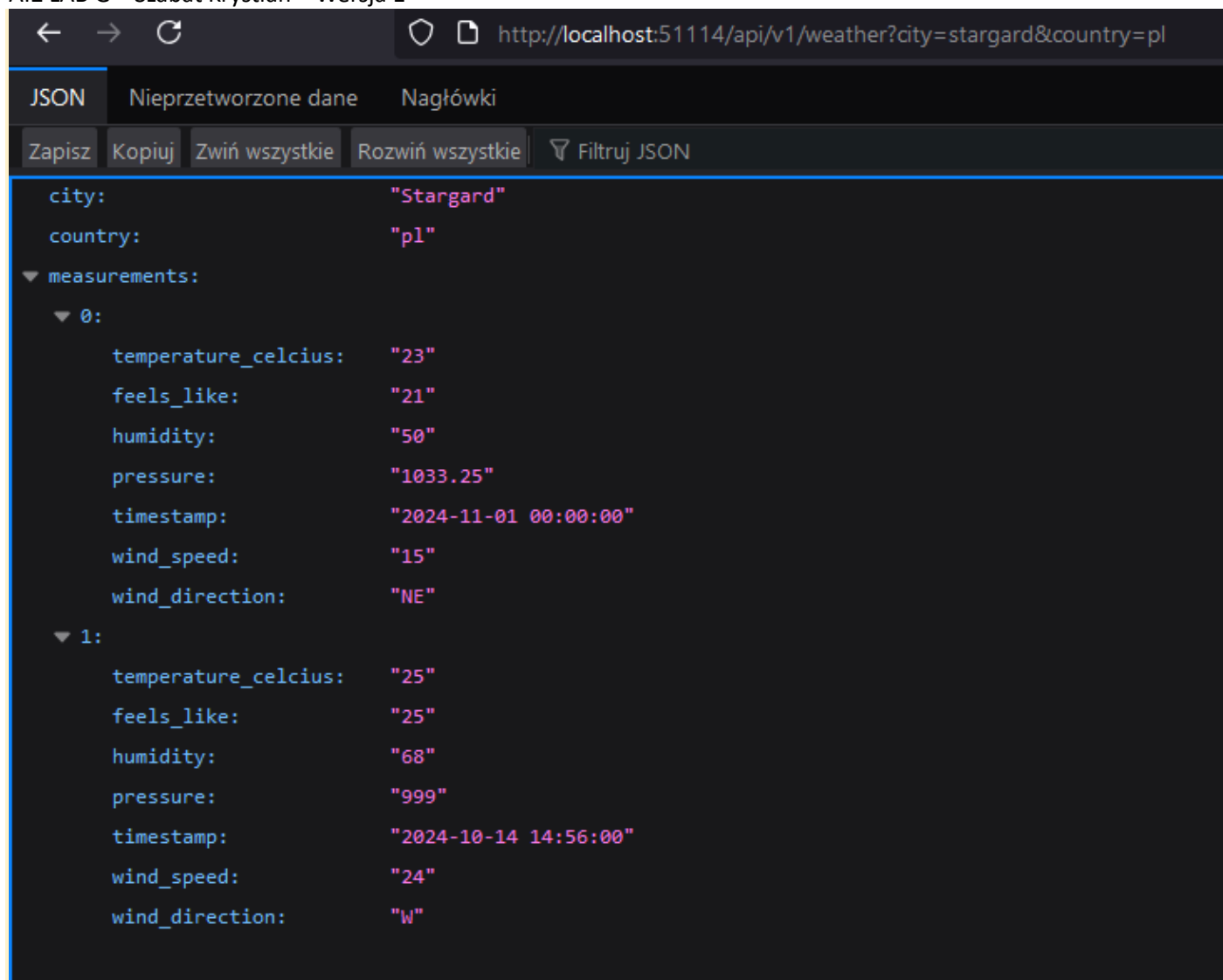


← → ↻ http://localhost:51114/api/v1/weather?city=szczecin&country=pl

JSON Nieprzetworzone dane Nagłówki

Zapisz Kopiuj Zwiń wszystkie Rozwiń wszystkie Filtruj JSON

```
{
  "city": "Szczecin",
  "country": "pl",
  "measurements": [
    {
      "temperature_celcius": "20",
      "feels_like": "20",
      "humidity": "50",
      "pressure": "1013.25",
      "timestamp": "2024-11-01 00:00:00",
      "wind_speed": "15",
      "wind_direction": "E"
    },
    {
      "temperature_celcius": "33",
      "feels_like": "59",
      "humidity": "33",
      "pressure": "333",
      "timestamp": "2024-10-24 10:05:00",
      "wind_speed": "33",
      "wind_direction": "E"
    }
  ]
}
```



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:

```

class WeatherApiController extends AbstractController
{
    #[Route('/api/v1/weather', name: 'app_weather_api')]
    public function index(
        #[MapQueryParameter(filter: \FILTER_VALIDATE_REGEXP, options: ['regexp' => '/^[a-zA-Z-]+$/'])] string $city,
        #[MapQueryParameter(filter: \FILTER_VALIDATE_REGEXP, options: ['regexp' => '/^[a-zA-Z]{2}$/'])] string $country,
        #[MapQueryParameter(filter: \FILTER_VALIDATE_REGEXP, options: ['regexp' => '/^(csv|json)$/'])] string $format,
        WeatherUtil $weatherUtil,
    ): Response {

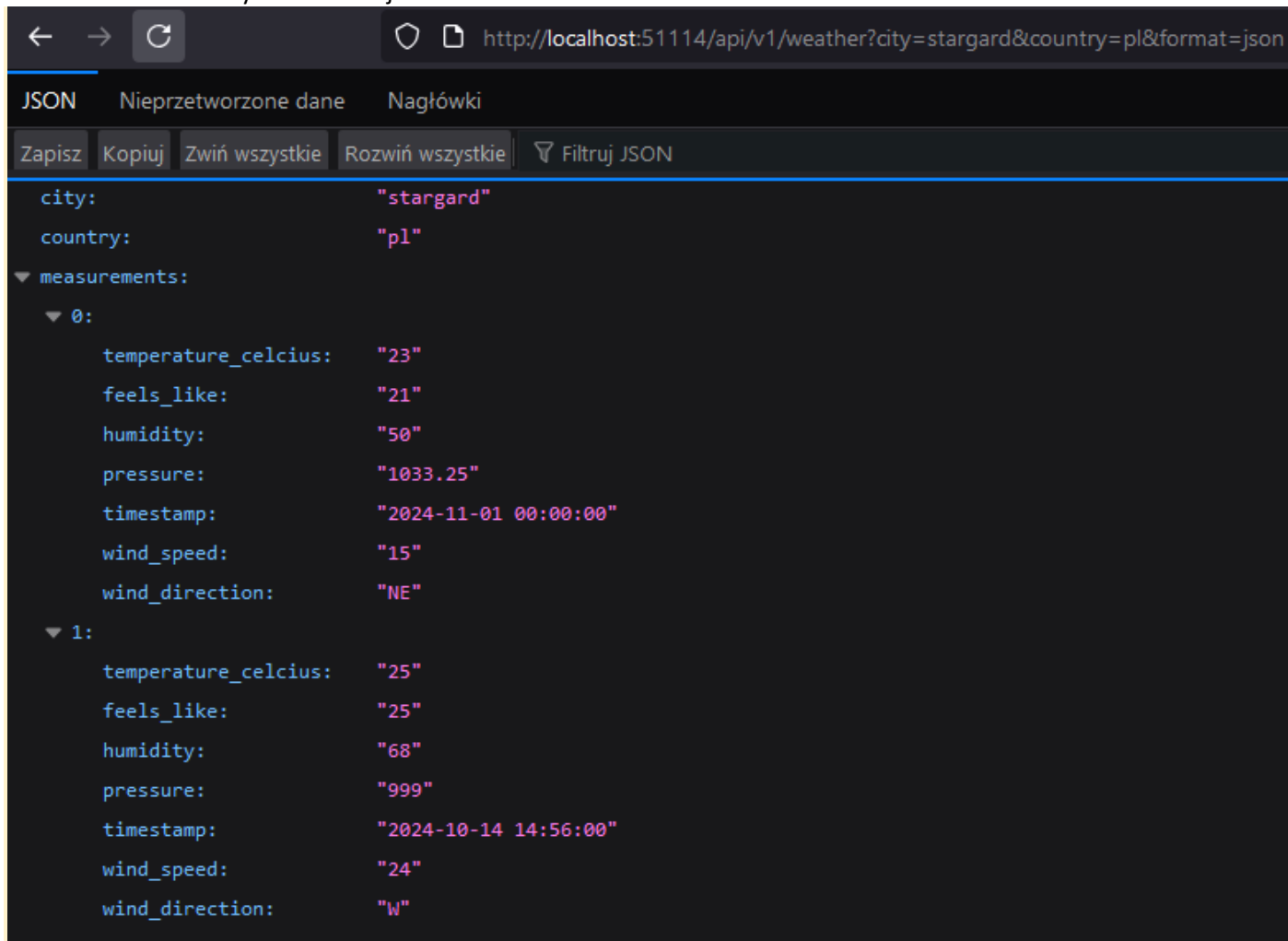
        $measurementEntries = $weatherUtil->getWeatherForCountryAndCity($country, $city);

        if ($format === 'json') {
            return $this->json(
                [
                    'city' => $city,
                    'country' => $country,
                    'measurements' => array_map(
                        fn(MeasurementEntry $m) => [
                            'temperature_celcius' => $m->getTemperatureCelcius(),
                            'feels_like' => $m->getFeelsLike(),
                            'humidity' => $m->getHumidity(),
                            'pressure' => $m->getPressure(),
                            'timestamp' => $m->getDateTime()->format('Y-m-d H:i:s'),
                            'wind_speed' => $m->getWindSpeed(),
                            'wind_direction' => $m->getWindDirection(),
                        ],
                        $measurementEntries
                    ),
                ],
            );
        } else if ($format === 'csv') {
            $csv = "city,country,temperature_celcius,feels_like,humidity,pressure,timestamp,wind_speed,wind_direction\n";
            foreach ($measurementEntries as $m) {
                $csv .= sprintf(
                    "%s,%s,%s,%s,%s,%s,%s,%s,%s\n",
                    $city,
                    $country,
                    $m->getTemperatureCelcius(),
                    $m->getFeelsLike(),
                    $m->getHumidity(),
                    $m->getPressure(),
                    $m->getDateTime()->format('Y-m-d H:i:s'),
                    $m->getWindSpeed(),
                    $m->getWindDirection()
                );
            }

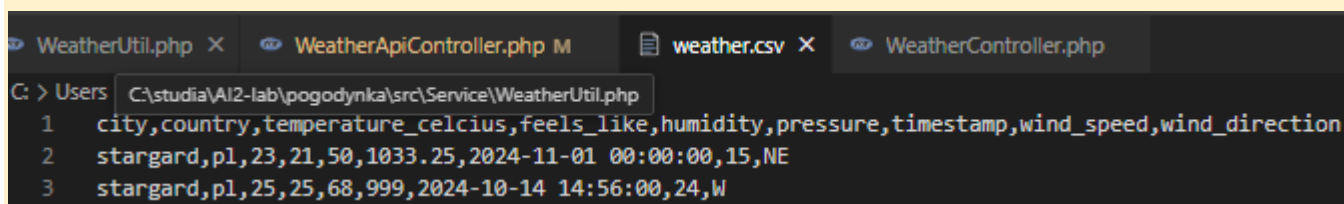
            return new Response($csv, 200, ['Content-Type' => 'text/csv']);
        }
    }
}

```

Wstaw zrzut ekranu otrzymanego z kontrolera JSONa:



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):**



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:

```

class WeatherApiController extends AbstractController
{
    #[Route('/api/v1/weather', name: 'app_weather_api')]
    public function index(
        #[MapQueryParameter(filter: \FILTER_VALIDATE_REGEXP, options: ['regexp' => '/^[a-zA-Z-]+$/'])] string $city,
        #[MapQueryParameter(filter: \FILTER_VALIDATE_REGEXP, options: ['regexp' => '/^[a-zA-Z]{2}$/'])] string $country,
        #[MapQueryParameter(filter: \FILTER_VALIDATE_REGEXP, options: ['regexp' => '/^(csv|json)$/'])] string $format,
        #[MapQueryParameter('twig')] bool $twig = false,
        WeatherUtil $weatherUtil,
    ): Response {

        $measurementEntries = $weatherUtil->getWeatherForCountryAndCity($country, $city);

        if ($format === 'json') {
            if ($twig === true) {
                return $this->render('weather_api/index.json.twig', [
                    'city' => $city,
                    'country' => $country,
                    'measurements' => $measurementEntries,
                ]);
            }

            return $this->json(
                [
                    'city' => $city,
                    'country' => $country,
                    'measurements' => array_map(
                        fn(MeasurementEntry $m) => [
                            'temperature_celcius' => $m->getTemperatureCelcius(),
                            'feels_like' => $m->getFeelsLike(),
                            'humidity' => $m->getHumidity(),
                            'pressure' => $m->getPressure(),
                            'timestamp' => $m->getDateTime()->format('Y-m-d H:i:s'),
                            'wind_speed' => $m->getWindSpeed(),
                            'wind_direction' => $m->getWindDirection(),
                        ],
                        $measurementEntries
                    ),
                ],
            );
        } else if ($format === 'csv') {
            if ($twig === true) {
                return $this->render('weather_api/index.csv.twig', [
                    'city' => $city,
                    'country' => $country,
                    'measurements' => $measurementEntries,
                ]);
            }

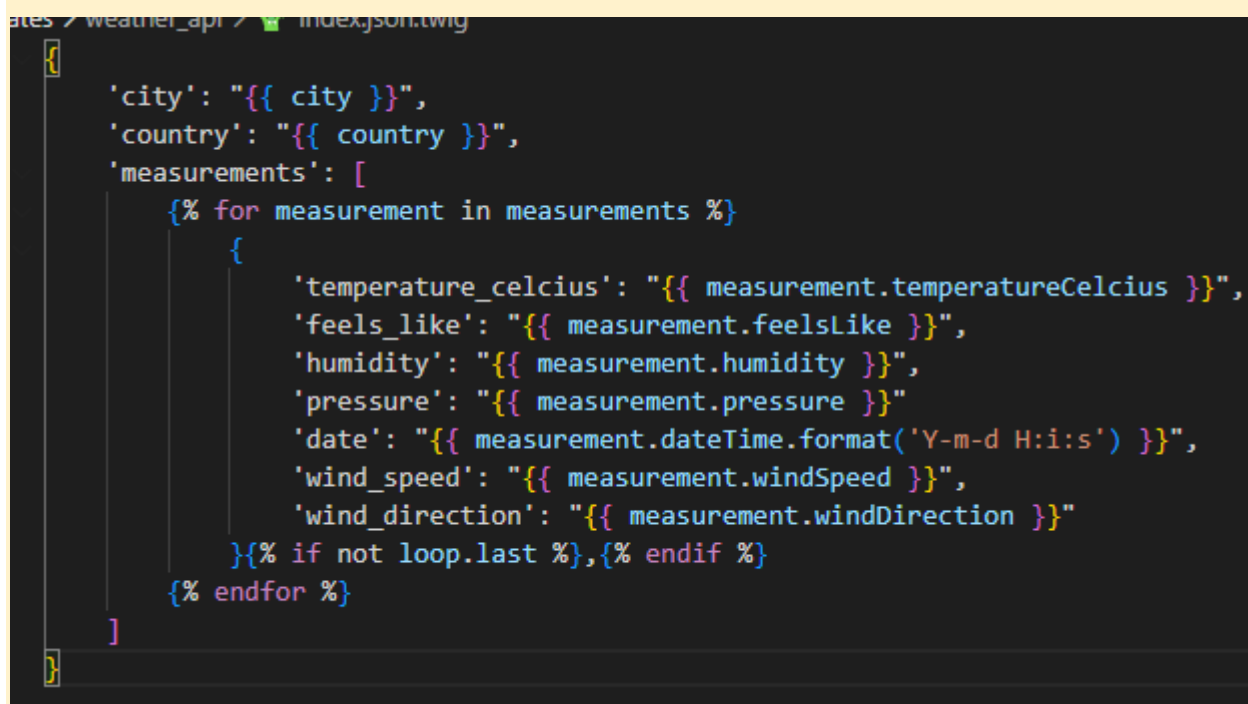
            $csv = "city,country,temperature_celcius,feels_like,humidity,pressure,timestamp,wind_speed,wind_direction\n";
            foreach ($measurementEntries as $m) {
                $csv .= sprintf(
                    "%s,%s,%s,%s,%s,%s,%s,%s,%s\n",
                    $city,
                    $country,
                    $m->getTemperatureCelcius(),
                    $m->getFeelsLike(),
                    $m->getHumidity(),
                    $m->getPressure(),
                    $m->getDateTime()->format('Y-m-d H:i:s'),
                    $m->getWindSpeed(),
                    $m->getWindDirection()
                );
            }

            return new Response($csv, 200, ['Content-Type' => 'text/csv']);
        }
    }
}

```

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

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

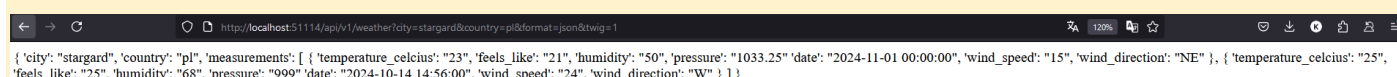


```

{
  'city': "{{ city }}",
  'country': "{{ country }}",
  'measurements': [
    {% for measurement in measurements %}
    {
      'temperature_celcius': "{{ measurement.temperatureCelcius }}",
      'feels_like': "{{ measurement.feelsLike }}",
      'humidity': "{{ measurement.humidity }}",
      'pressure': "{{ measurement.pressure }}"
      'date': "{{ measurement.dateTime.format('Y-m-d H:i:s') }}",
      'wind_speed': "{{ measurement.windSpeed }}",
      'wind_direction': "{{ measurement.windDirection }}"
    }{% if not loop.last %},{% endif %}
    {% endfor %}
  ]
}

```

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).



```

{ 'city': 'stargard', 'country': 'pl', 'measurements': [ { 'temperature_celcius': '23', 'feels_like': '21', 'humidity': '50', 'pressure': '1033.25' 'date': '2024-11-01 00:00:00', 'wind_speed': '15', 'wind_direction': 'NE' }, { 'temperature_celcius': '25', 'feels_like': '25', 'humidity': '68', 'pressure': '999' 'date': '2024-10-14 14:56:00', 'wind_speed': '24', 'wind_direction': 'W' } ] }

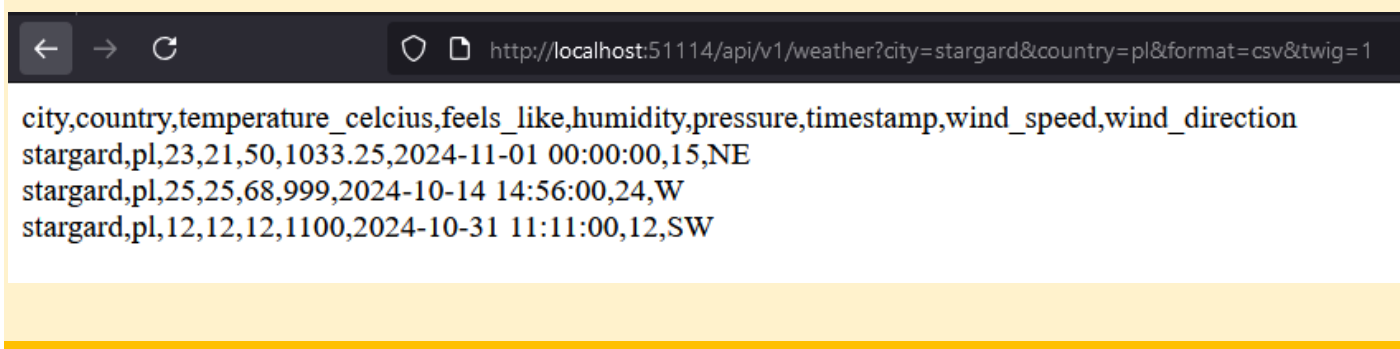
```

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:**

```
city,country,temperature_celcius,feels_like,humidity,pressure,timestamp,wind_speed,wind_direction
{% for measurement in measurements %}
    {"\n"|n12br}}
    {{- city -}},
    {{- country -}},
    {{- measurement.temperatureCelcius -}},
    {{- measurement.feelsLike -}},
    {{- measurement.humidity -}},
    {{- measurement.pressure -}},
    {{- measurement.dateTime.format('Y-m-d H:i:s') -}},
    {{- measurement.windSpeed -}},
    {{- measurement.windDirection -}}
{% 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).



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()`:

```
public function getFahrenheit(): ?string
{
    $fahrenheit = number_format(($this->temperature_celcius * 9 / 5) + 32), 2);
    return $fahrenheit;
}
```

You, 2 minutes ago • Uncommitted changes

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

city:	"stargard"
country:	"pl"
▼ measurements:	
▼ 0:	
temperature_celcius:	"23"
feels_like:	"21"
humidity:	"50"
pressure:	"1033.25"
timestamp:	"2024-11-01 00:00:00"
wind_speed:	"15"
wind_direction:	"NE"
temperature_fahrenheit:	"73.40"
▼ 1:	
temperature_celcius:	"25"
feels_like:	"25"
humidity:	"68"
pressure:	"999"
timestamp:	"2024-10-14 14:56:00"
wind_speed:	"24"
wind_direction:	"W"
temperature_fahrenheit:	"77.00"
▼ 2:	
temperature_celcius:	"12"
feels_like:	"12"
humidity:	"12"
pressure:	"1100"
timestamp:	"2024-10-31 11:11:00"
wind_speed:	"12"
wind_direction:	"SW"
temperature_fahrenheit:	"53.60"

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 można użyć metody `getFahrenheit()`, ponieważ celem testu jest sprawdzenie czy zwraca ona poprawne wartości, w takim przypadku porównywalibyśmy ją z samą sobą, żaden błąd nie zostałby wykryty.

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`:

```
<?php

namespace App\Tests\Entity;

use App\Entity\MeasurementEntry;
use PHPUnit\Framework\TestCase;

class MeasurementEntryTest extends TestCase
{
    public function testGetFahrenheit(): void
    {
        $measurement = new MeasurementEntry();

        // Test 1: celcius = 0, fahrenheit = 32
        $measurement->setTemperatureCelcius(0);
        $this->assertEquals(32, $measurement->getFahrenheit());

        // Test 2: celcius = -100, fahrenheit = -148
        $measurement->setTemperatureCelcius(-100);
        $this->assertEquals(-148, $measurement->getFahrenheit());

        // Test 3: celcius = 100, fahrenheit = 212
        $measurement->setTemperatureCelcius(100);
        $this->assertEquals(212, $measurement->getFahrenheit());
    }
}
```

Wklej rzut ekranu wywołania i wyniku testów:

```
PS C:\studia\AI2-lab\pogodynka> php .\bin\phpunit
PHPUnit 9.6.21 by Sebastian Bergmann and contributors.
```

```
Testing
```

```
1 / 1 (100%)
```

```
Time: 00:00.269, Memory: 6.00 MB
```

```
OK (1 test, 3 assertions)
```

```
PS C:\studia\AI2-lab\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`:


```
<?php

namespace App\Tests\Entity;

use App\Entity\MeasurementEntry;
use PHPUnit\Framework\TestCase;

class MeasurementEntryTest extends TestCase
{
    public function dataGetFahrenheit(): array
    {
        return [
            ['0', 32],
            ['-100', -148],
            ['100', 212],
            ['37', 98.6],
            ['100.4', 212.72],
            ['98.6', 209.48],
            ['5', 41],
            ['0.1', 32.18],
            ['-0.1', 31.82],
            ['-5', 23],
        ];
    }

    /**
     * @dataProvider dataGetFahrenheit
     */
    public function testGetFahrenheit($celsius, $expectedFahrenheit): void
    {
        $measurement = new MeasurementEntry();

        $measurement->setTemperatureCelcius($celsius);
        $this->assertEquals($expectedFahrenheit, $measurement->getFahrenheit());
    }
}
```

Wklej zrzut ekranu wywołania i wyniku testów:

```
PS C:\studia\AI2-lab\pogodynka> php .\bin\phpunit
PHPUnit 9.6.21 by Sebastian Bergmann and contributors.

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

Time: 00:00.016, Memory: 6.00 MB

OK (10 tests, 10 assertions)
PS C:\studia\AI2-lab\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/szvbvtk/ai2-pogodynka/tree/lab-g>

PODSUMOWANIE

W kilku zdaniach podsumuj zdobyte podczas tego laboratorium umiejętności.

Podczas laboratorium nauczyłem się jak tworzyć API w Symfony oraz jak generować odpowiedzi w różnych formatach, w tym przy użyciu szablonów TWIG.

Kolejną zdobytą umiejętnością jest tworzenie testów jednostkowych oraz testowanie wielu przypadków dzięki `dataGetFahrenheit()`

Zweryfikuj kompletność sprawozdania. Utwórz PDF i wyślij w terminie.