CurrencyRates

Architektura

Zastosowano architekturę ‘Onion Architecture’. Architektura oparta jest na interfejsach. Ich aktualne implementacje mogą zostać w razie potrzeby łatwo zastąpione. Na przykład w razie zmiany dostawcy danych o kursach lub w razie zmiany sposobu lokalnego przechowywania danych. Onion architecure ułatwia testowanie przez brak zależności pomiędzy warstwami projektu.

Endpointy

/api/rates z metodą GET umożliwiającą pobranie danych walutowych. Poniżej przykładowe wywołanie

```
api/rates?apiKey=API_KEY

Body:

{
        "startDate": "2019-06-01",
        "endDate": "2020-05-17",
        "currencyCodes":
        {
                "USD": "EUR"
        }
}
```

/api/key z metodą GET umożliwiającą pobranie nowego klucza API

Baza danych

Do cachowania danych wykorzystano bazę danych SQLServer. Wykorzystany ORM to EntityFrameworkCore. SQLServer używany jest także to przechowywania wygenerowanych kluczy API.

Dostęp do bazy danych reprezentowany jest przez interfejsy `ICurrencyRepository` opdowiedzialny za dane walutowe oraz `IApiKeyRepository` reprezentujący dostęp do zasobu przechowującego klucze API.

ApiKey

W celu zabezpieczenia API kluczem został zaimplementowany `ApiKeyAuthenticationHandler.` Dzięki temu kontrolery lub metody z atrybutem `[Authorize]` wymagają klucza API przesłanego jako parametr postaci:

```
api/rates?apiKey=API_KEY
```

Kontener DI

Wykorzystano domyślny kontener .net core.

Logowanie

Do logowania wykorzystano bibliotekę Serilog. Dane logowane są do plików w katalogu Logs. Aby zalogować informację o każdym zapytaniu API zaimplementowany został Middleware (`LoggingMiddleware`).

Benchmark

Do przeprowadzenia testów wydajnościowych użyto bibliotekę dotnetbenchmark. Porównano rozwiązanie wykorzystujące cachowanie danych w lokalnej bazie z rozwiązaniem bez cachowania.

Parametry wejściowe:

- startDate oraz endDate: losowe daty z przedziału 2019-01-01 – 2020-05-16, startDate < endDate
- CurrencyCodes: USD -> EUR
- Liczba iteracji: 100

Poniżej przedstawiono porównanie średniego czasu odpowiedzi dla rozwiązanie z cache (WithCache) i bez cache (WithoutCache)

```
BenchmarkDotNet=v0.12.1, OS=Windows 10.0.18362.836 (1903/May2019Update/19H1)
Intel Core i7-4702MQ CPU 2.20GHz (Haswell), 1 CPU, 8 logical and 4 physical
cores
.NET Core SDK=3.1.100
  [Host]     : .NET Core 3.1.0 (CoreCLR 4.700.19.56402, CoreFX
4.700.19.56404), X64 RyuJIT  [AttachedDebugger]
  DefaultJob : .NET Core 3.1.0 (CoreCLR 4.700.19.56402, CoreFX
4.700.19.56404), X64 RyuJIT
```

Method	Mean	Error	StdDev	Median
WithoutCache	683.904 ms	157.0129 ms	437.6892 ms	537.808 ms
WithCache	2.162 ms	0.2079 ms	0.5931 ms	2.019 m

Średnia czasu odpowiedzi to 684ms bez zastosowania cache i 2ms z wykorzystaniem cache.