Running average

This document describes a backend homework challenge for candidates applying for positions within Maersk's Service Delivery platform.

The goal is to see:

- How you approach a problem
- How you design, structure, implement, and deliver a complete solution that is adapted to the proposed problem below

This will be followed by a discussion with some of the software engineers in Service Delivery's hiring team.

Handing in your solution

- Your code and other deliverables must be provided as a link to a private Git repository (VSTS, GitLab, GitHub, or Bitbucket)
- Your solution must only be accessible to you and us; please make sure it is not available for a wider audience, especially not publicly. We would like to reuse the challenge - please help us keeping it fair!

The problem

Your assignment is to build an ASP.NET Core web API for an internal dashboard application to track container transport prices for specific geographical routes, referred to as *voyages*. More specifically, your API needs an endpoint that returns the last 10 prices for containers booked on a given voyage and an endpoint through which you can register a new booking price.

The application should support at least 3 different currencies of your choice. You don't need to connect to an online service to get the latest currency rates - it is fine to use hard-coded exchange rates.

Further, all data should be kept in memory.

Endpoints

The API should have the following two endpoints:

- `[POST] UpdatePrice(string voyageCode, decimal price, Currency currency, DateTimeOffset timestamp)`
  - Example: `UpdatePrice("451S", 109.5, Currency.Gbx, DateTimeOffset.Now)`

- `[GET] GetAverage(string voyageCode, Currency currency)`
  - Example: `GetAverage("451S", Currency.Gbx)` --> 152.35

We expect a lot of traffic, particularly for reads, so performance is important to keep in mind.

Your solution

Your delivery should:

- Include a fully functional solution built using C#, ASP.NET Core, and .NET Core
- Include the complete code needed to execute the solution
- Include a suite of unit tests covering key components of your solution
- Contain logging functionality
- If you use any third-party libraries, these must be referenced via NuGet
- Contain a Readme file explaining how to build, test, and run your solution locally, plus any additional details (e.g. architectural decisions) you might find relevant for us to know

Your solution should be self-contained and all data should be stored **in memory** for the sake of simplicity. No external infrastructure should be needed to run the solution.

The requirements intentionally leave a lot of design decisions up to you, so use your solution to showcase how you would approach the described problem from both a (technical) design and implementation point of view.

Good Luck! We look forward to reviewing your solution.