VI/COMPANY

VI Company - Backend Assessment

Thank you for taking the time for the VI Company Developer Challenge! Please make yourself comfortable for a fun and challenging test. To make the process as smooth as possible, try to limit distractions.

Our aim is to get a first impression of your work as a developer. We will not only focus on the results; we are also very interested in your reasoning behind choosing certain solutions. Don't be afraid to approach problems in your own way or ask for help when you get stuck. This is something that we value in our daily work as well.

It should take about 2 hours to complete all three objectives. Did you spend less time? That’s fine! Do you need more time? That’s fine too, but please don’t spend more than 3 hours on the objectives. Also, keep in mind that the unit tests are there to validate your implementations, so they are not meant to be changed. But enough about that—let’s get to the objectives!

Introduction

Since VI Company works in the financial sector, our intake test is inspired by this domain. The project resembles our real projects. The website in the intake test displays a user’s portfolio dashboard and the latest quotes for certain financial instruments. The solution consists of three projects:

- **Domain:** contains all models, repositories and services;
- **Frontend:** this is the website and contains viewmodels and views;
- **Tests:** the unit tests for our domain logic.

Objective 1 – Implement the constructor of Portfolio

The constructor method of **Portfolio** should transform a user's transactions into a **Position** instance for each instrument and a cash position. An instrument position consists of an instrument identifier (ISIN), a weighted average purchase price, and the total quantity held. The **startCash** parameter represents the amount of money the user had before initiating the transactions. The cash position represents the total remaining amount after the user's buy and sell transactions.

Example: Consider the following situation: A user buys two stocks at 10 euro and later buys 5 at 20 euro. The average purchase price of the bought stock may then be calculated as follows:

$$\frac{(2 * 10) + (5 * 20)}{7} = 17{,}14\, EUR$$

Your implementation is verified by the unit tests in PortfolioTests class.

Objective 2 – Replace DummyQuotesRepository with live data from Tickly

While the DummyQuotesRepository serves its purpose during development, we want to use data from our application, https://tickly.vicompany.io/, for historical quote data. Please review the API and use the historical data that can be retrieved from it. Feel free to use external packages for accessing a REST API. Please transform the data from the Tickly format into Quote object instances.

You can validate your implementation by running the website and viewing the quotes. Pay attention to the difference in timestamp between Tickly and our application. We want the **DateTime** to be in accordance with the ISO 8601 format.

Objective 3 – Refactor the solution

The code in this solution has room for improvement. Please identify any areas in the solution that can be refactored, and be sure to explain your reasoning in your reply email or Git commit. If time allows, you can refactor multiple aspects; however, we still want to understand the reasoning behind each refactor and why you believe the change is important.

You may refactor anything, such as naming, structure, layout, or anything else that catches your attention. However, please refrain from changing the unit test setup, as this would interfere with objectives 1 and 2. Feel free to provide suggestions for improvements on those as well!

Final note: we would love to see some form of work log for your assignment, including what changes you made, why you made them, and in what order. Good luck! :)

Some development tips

- Do you want to add new unit tests? Here is how we do it:
  https://www.vicompany.nl/en/insights/mastering-the-subtle-art-of-unit-testing
- Do you want to know how we write clean code?
  https://www.vicompany.nl/en/insights/clean-code-why-you-should-prefer-to-keep-it-clean
- At VI company we all use Resharper by JetBrains. The code also features .DotSettings and Settings.StyleCop which are used to show you our code style when using resharper and plugins.
  https://www.jetbrains.com/resharper/download/
  https://plugins.jetbrains.com/plugin/11619-stylecop-by-jetbrains
- Recently we are growing fonder of using immutable data approaches and functional techniques.