**Code Refactoring: Strategy Design Pattern Implementation**

**Introduction**

In this document, I will explain the design pattern we implemented to refactor our code. The primary goal of this refactor was to improve the maintainability and clarity of the code. By applying the Strategy Pattern, we separated the different technical indicators into individual strategy classes, making the code easier to extend, debug and test.

**Strategy Design Pattern**

The key advantage of using the Strategy Pattern is that it allows a class to change its behavior at runtime without modifying its code, simply by selecting a different strategy. This was an ideal choice for our case because we have multiple algorithms performing similar tasks, such as calculating different technical indicators (RSI, SMA, EMA), and we wanted to be able to switch between them dynamically, without changing the main logic with conditionals or hardcoded implementations.

**Implementation of the Strategy Pattern**

We created individual strategy classes for each technical indicator. Each strategy encapsulates its own calculation logic, making the code easier to maintain.

We also defined a common interface called TechnicalIndicatorStrategy, which includes a calculate method.

All the indicator classes (RSI, SMA, EMA) implement this interface. This ensures that each strategy following a consistent structure, provides flexibility to easily add new indicators in the future without changing everything. If a new indicator is needed, we can create a new class that will implement the calculate method, and the rest of the application stays unaffected.

With this refactor, we achieved a separation of concerns. Each technical indicator's calculation logic is now contained in its own strategy class.