Refactoring Code with Software Design Patterns

Implemented Design Patterns

Template Pattern

The scrapers and analyzers in the application followed similar workflows, such as starting processes, handling data, and generating outputs. By implementing the Template Pattern, we created a shared structure for these workflows while allowing specific steps to be customized. A base class defined the overarching process, while subclasses implemented their unique steps, such as data fetching or analysis.

Decorator Pattern

In both scrapers and API services, the Decorator Pattern was used to add extra functionalities dynamically. For example, logging and multiprocessing capabilities were added to scrapers, while caching mechanisms were added to API services. This approach allowed us to enhance the functionality of these components without altering their core logic.

Factory Pattern

API services required the creation of different types of service objects for tasks like fetching data about issuers, companies, or stocks. To simplify and centralize this process, we implemented the Factory Pattern. A factory class now provides the correct service object based on the type requested, making the code more organized and easier to extend.

Singleton Pattern

Some API services are resource-intensive, and creating multiple instances of these services would have been inefficient. By applying the Singleton Pattern, we ensured that only one instance of each service exists during the application’s runtime. This not only conserved resources but also improved consistency across the application.

Benefits of Refactoring

The refactored code is now more maintainable, as shared processes have been centralized using patterns like Template and Factory. This reduces redundancy and makes updates more straightforward. Scalability has also been enhanced, with decorators allowing new functionalities to be added seamlessly without altering existing code. Efficiency has been significantly improved, with the Singleton Pattern and caching mechanisms preventing unnecessary object creation and optimizing performance.

Conclusion

The refactoring process has significantly improved the structure and performance of the application. By using design patterns, we tackled inefficiencies, reduced code duplication, and made the system more adaptable to future requirements. This effort ensures the application remains reliable, efficient, and easier to maintain as it evolves.