STOCK MANAGEMENT

Northeastern University | 2024

Team Members

- Gokul Jayavel
- Geetha Parthasarathy
- Nethra Viswanathan
- Harshan Goodwin Hector
- Evita Paul
- Chaman
- Prasanna Balaji

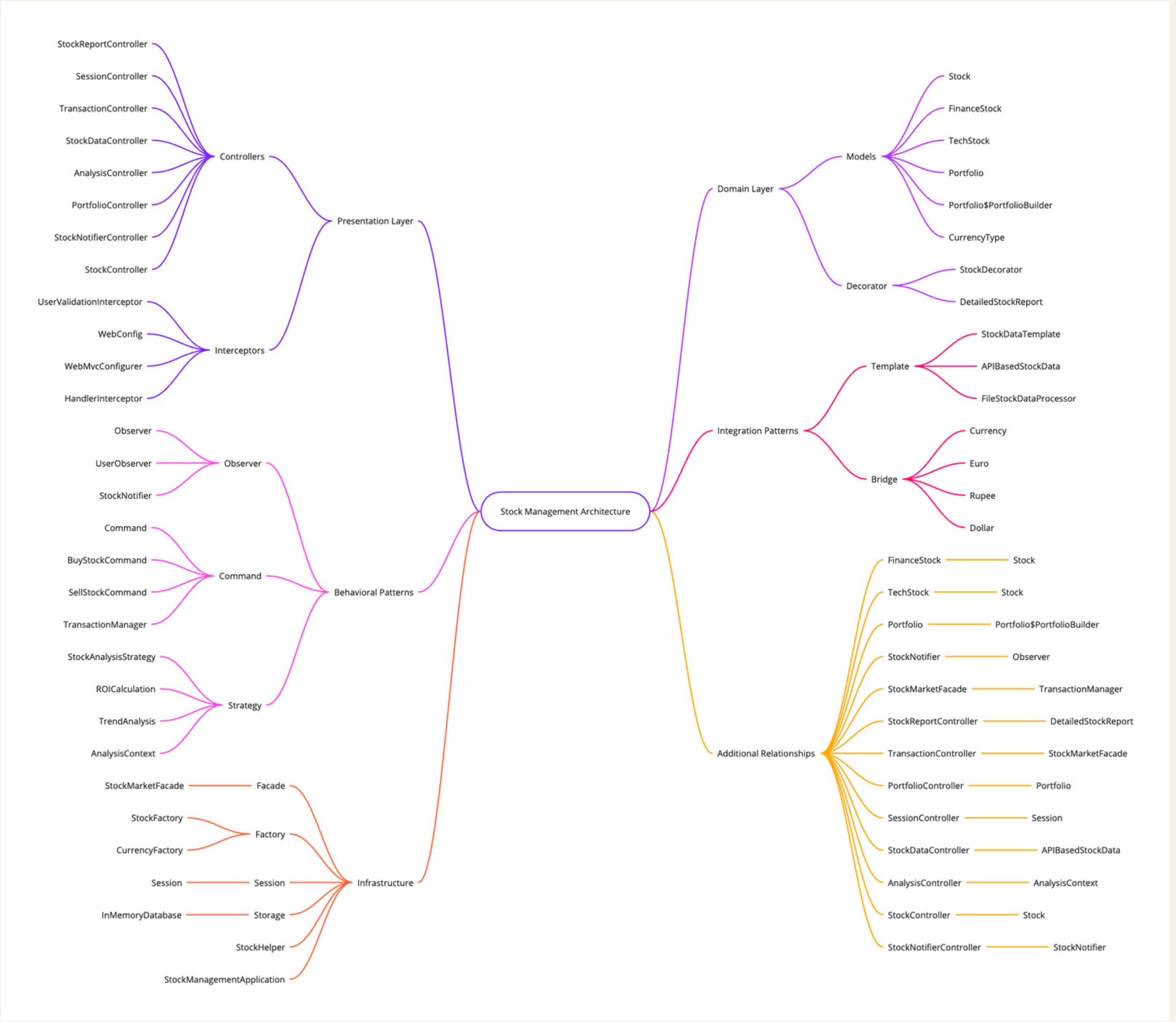
Overview

Patterns Implemented:

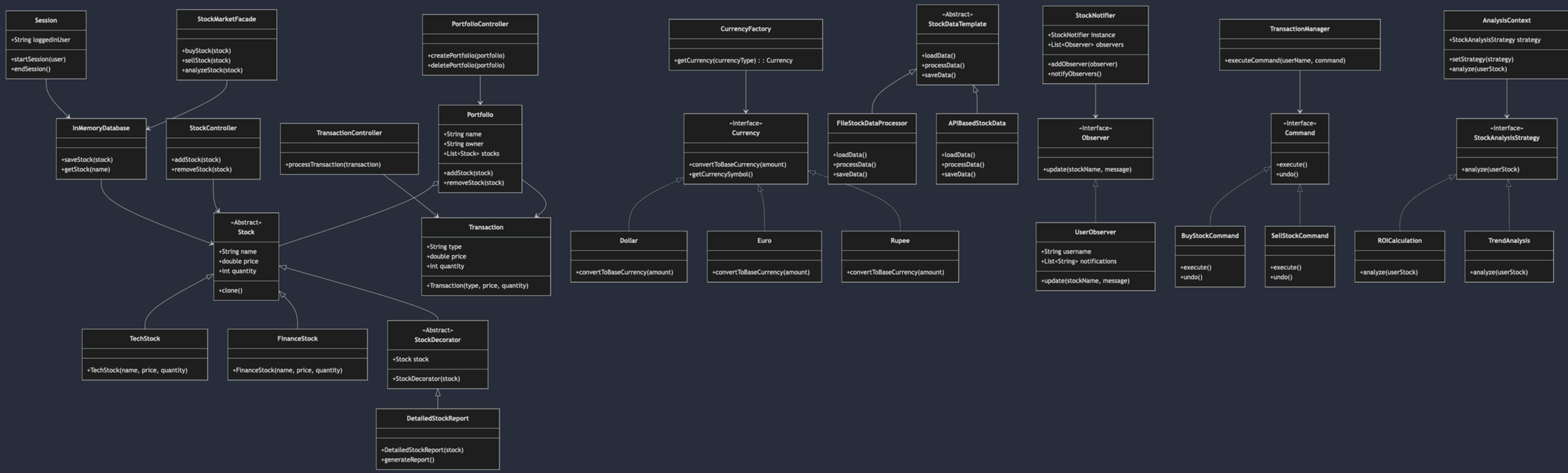
- **Decorator**: For dynamic reporting enhancements (e.g., detailed stock reports).
- **Bridge**: To handle multiple currency types (e.g., Dollar, Euro).
- **Template**: For reusable stock data processing workflows.
- **Facade**: To simplify complex stock market API interactions.
- **Observer**: To notify users of stock updates dynamically.
- **Command**: For executing buy/sell stock transactions.
- **Factory**: To create stock-related objects.
- **Strategy**: For implementing dynamic stock analysis techniques.

Controllers and Models:

- **Controllers**: Handle user requests and map them to business logic, enabling seamless interaction with the application (e.g., Portfolio, Transaction, Stock APIs).
- **Models**: Represent domain-specific entities like Stock, Portfolio, and CurrencyType, ensuring data consistency and clarity.


Stock Management Architecture
Presentation Layer
Controllers
StockReportController
SessionController
TransactionController
StockDataController
AnalysisController
PortfolioController
StockNotifierController
StockController
Interceptors
UserValidationInterceptor
WebConfig
WebMvcConfigurer
HandlerInterceptor
Behavioral Patterns
Observer
Observer
UserObserver
StockNotifier
Command
Command
BuyStockCommand
SellStockCommand
TransactionManager
Strategy
StockAnalysisStrategy
ROICalculation
TrendAnalysis
AnalysisContext
Infrastructure
Facade
StockMarketFacade
Factory
StockFactory
CurrencyFactory
Session
Session
Storage
InMemoryDatabase
StockHelper
StockManagementApplication
Domain Layer
Models
Stock
FinanceStock
TechStock
Portfolio
Portfolio$PortfolioBuilder
CurrencyType
Decorator
StockDecorator
DetailedStockReport
Integration Patterns
Template
StockDataTemplate
APIBasedStockData
FileStockDataProcessor
Bridge
Currency
Euro
Rupee
Dollar
Additional Relationships
FinanceStock
Stock
TechStock
Stock
Portfolio
Portfolio$PortfolioBuilder
StockNotifier
Observer
StockMarketFacade
TransactionManager
StockReportController
DetailedStockReport
TransactionController
StockMarketFacade
PortfolioController
Portfolio
SessionController
Session
StockDataController
APIBasedStockData
AnalysisController
AnalysisContext
StockController
Stock
StockNotifierController
StockNotifier

Class Diagram



Factory Pattern

A creational design pattern that provides an interface for creating objects without specifying their exact class.

Implemented it to dynamically create stock objects (e.g., TechStock, FinanceStock)
based on type

- StockFactory: Singleton class for creating stock instances.
- Stock: Abstract class defining the base structure.
- TechStock & FinanceStock: Concrete implementations created by the factory.

Command Pattern

A behavioral design pattern that encapsulates a request as an object, allowing parameterization, queuing, and reversible operations.

Implemented it to handle stock transactions (buy/sell) with undo functionality for managing transaction history.

- **Command**: Interface defining execute and undo methods.
- **BuyStockCommand & SellStockCommand**: Concrete classes for stock operations.
- **TransactionManager**: Executes and tracks commands

Observer Pattern

A behavioral design pattern where objects (observers) subscribe to and get notified of changes in another object (subject).

Implemented it to notify users about stock updates in real-time.
Tracks user subscriptions and ensures they are notified of stock price or availability changes dynamically.

- **Observer: Interface defining the update method.**
- **UserObserver: Concrete class for user-specific notifications.**
- **StockNotifier: Singleton class that manages and notifies observers.**

Template Method Pattern

A behavioral design pattern that defines the skeleton of an algorithm in a method, deferring steps to subclasses.

Implemented it to standardize data processing workflows for stock data from various sources.
Provides a reusable structure for processing stock data while allowing source-specific customizations

01 **StockDataTemplate**: Abstract class defining the algorithm's skeleton

02 **APIBasedStockData** : Concrete classes implementing specific data fetching logic

03 **FileStockDataProcessor** : Concrete classes implementing specific data-fetching logic

Decorator Pattern

A structural design pattern that dynamically adds behavior to an object without altering its structure.

Implemented it to enhance stock reports with additional details.Enhances stock reporting functionality without modifying the original Stock class.

- **StockDecorator**: Abstract class defining the base decorator.
- **DetailedStockReport**: Concrete decorator adding advanced report details.
- **Stock**: Core component being decorated.

Strategy Pattern

A behavioral design pattern that defines a family of algorithms and allows them to be interchangeable.

Implemented it for various stock analysis methods (e.g., ROI calculation, trend analysis).

- StockAnalysisStrategy: Interface defining the analysis method.
- ROICalculation & TrendAnalysis: Concrete classes implementing specific analysis strategies.
- AnalysisContext: Manages and executes selected strategies.

Facade Pattern

A structural design pattern that provides a simplified interface to a complex subsystem.

StockMarketFacade: Provides a high-level interface for transaction operations.

Implemented it to simplify access to stock market operations like buying, selling, and checking stock details.

TransactionManager: Underlying subsystem used by the facade.

Bridge Pattern

A structural design pattern that separates abstraction from implementation, allowing them to evolve independently.

Implemented it to handle different currency conversions (e.g., USD, EUR, INR) independently from stock logic.

- Currency: Interface defining currency operations.
- Dollar, Euro, Rupee: Concrete implementations of Currency.
- CurrencyFactory: Creates currency objects.

Decouples stock functionality from currency operations, enabling easy extension for new currencies.

Thank you!