Software architecture and design

Sebastian

This document presents:

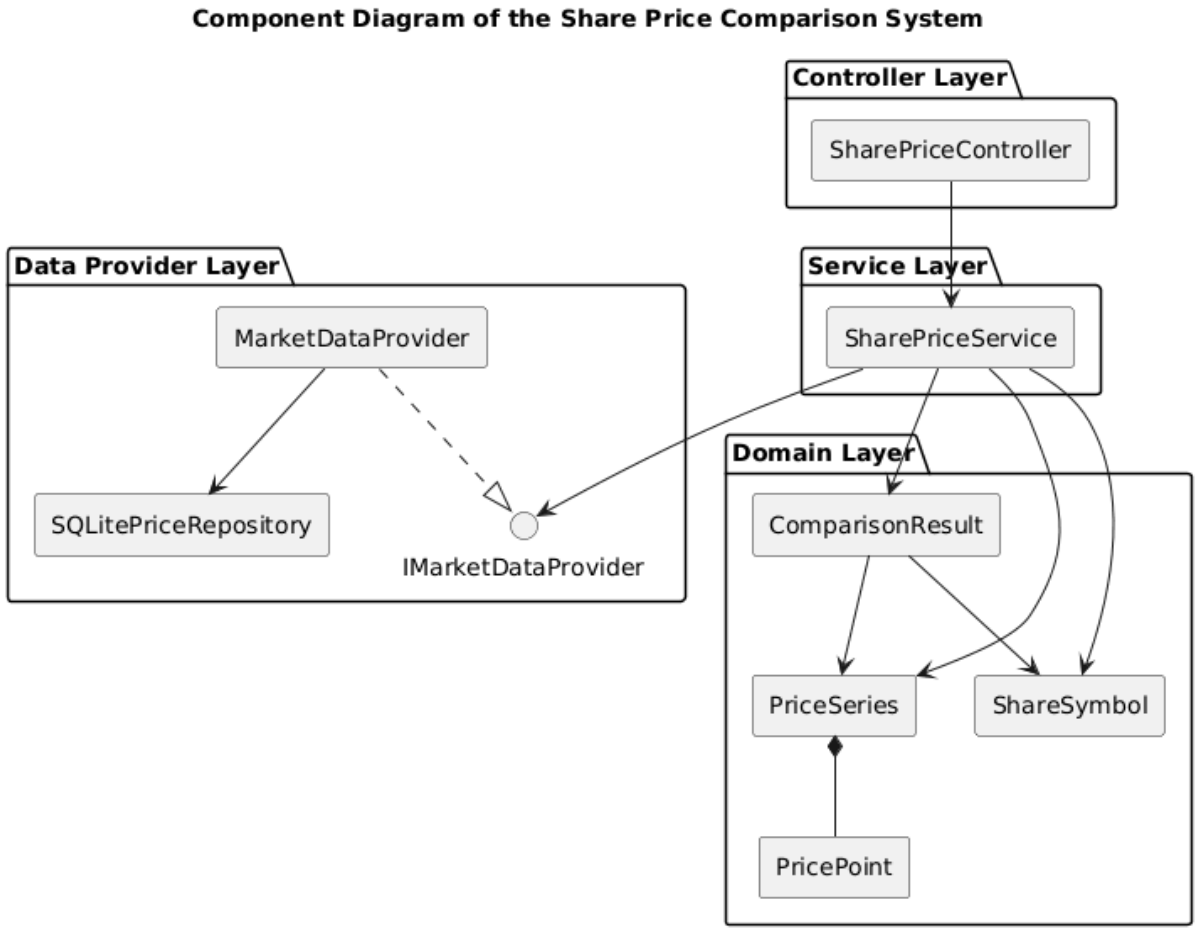
Architectural Pattern

The application uses a Layered Architecture Pattern implemented in plain Java. The main goal of this architecture is to separate responsibilities so that user interaction, business logic, data retrieval, and persistence remain independent and easy to maintain.

Layer	The use of the system	Why we are using this layer	Risks It Prevents + Benefits
Controller Layer	Handles user requests and interaction flow. Performs basic validation and delegates processing to the Service layer. Returns structured output	We use a Controller layer to separate presentation/request-handling logic from business rules. This ensures that changes to user input formats, routing, or interface behaviour do not affect core calculation logic.	Reduces duplication of validation logic. Improves maintainability by isolating presentation concerns from application rules.
Service Layer	Contains the core business logic of the system. Enforces rules such as maximum two-year date range. Coordinates data retrieval from the provider. Performs comparison calculations	The Service layer centralises all application rules and comparison logic. This makes the system scalable and easier to test. Any future enhancement like moving averages can be added here without modifying controllers or data access code. It also ensures consistent enforcement of business rules across all features.	Reduces tight coupling between data retrieval and calculation logic. Improves testability because the service can be tested independently using mock providers.
Data Provider Layer	Retrieves share price data from external sources and/or local SQLite storage. Abstracts the source of data behind an interface. Manages persistence for offline access.	We use a Data Provider layer to isolate integration logic (API calls and SQLite operations) from the rest of the system. This allows us to switch data sources or implement offline fallback without affecting business logic. It ensures clean separation between external infrastructure and internal application logic.	Prevents external API code leaking into business logic. Improves modularity and adaptability.
Domain Layer	Defines core data models like PricePoint, PriceSeries, ShareSymbol, and ComparisonResult. Contains no framework or database-specific logic.	The Domain layer provides a stable representation of the system's core concepts. All layers operate using these models, ensuring consistency and reducing conversion errors. Because domain objects are independent of frameworks and databases, they are reusable and easy to maintain.	Reduces coupling to frameworks or persistence technologies. Improves clarity of system design.

This layered architecture divides the application into small constituent parts which can help with dealing with potential problems that may occur in the future, low coupling between components. By dividing responsibilities into Controller, Service, Data Provider, and Domain layers, the system achieves improved maintainability, scalability, and testability. This design supports both current requirements such as share price retrieval and comparison and future extensibility, including additional analytical features or alternative data sources.

Component Diagram



Relationships:

The relationship between each component here are as follows:

- The use of dependency where one component relies on the other is shown with controller using service
- The use of a realization component where one component implements an interface is seen when the marketdataprovider is implementing Imarketdataprovide