Design Patterns Used in the Application

1. Singleton Pattern

- **Where It's Used:** The DatasourceConfig class in the data-access-api package demonstrates the Singleton pattern. The @Bean annotation in Spring ensures that the DataSource object is instantiated as a singleton within the application context.
- **Why We Used It:** We needed a single, shared database connection pool to manage all database queries efficiently, reducing resource overhead and ensuring thread safety.

2. Adapter Pattern

- **Where It's Used:** The DTO conversion methods, such as IndexDto.fromEntity and NewsSentimentsDto, act as Adapters between entities and DTOs. They transform database entities into API-friendly DTOs.
- **Why We Used It:** This allows decoupling of the database structure from the frontend requirements, enabling us to change one without affecting the other.

3. MVC Pattern

- **Where It's Used:** The overall structure of each microservice follows the MVC pattern:
    - **Model:** Entities like Index, PerformanceMetrics, and NewsSentiments.
    - **View:** Represented indirectly by the JSON responses sent to the frontend.
    - **Controller:** Controllers like IndexController, MetricsController, and LSTMPredictionController handle requests and responses.
    - **Service Layer:** Bridges the Controller and Repository layers, implementing the business logic.
- **Why We Used It:** MVC ensures a clean separation of concerns, making the codebase easier to maintain and extend.

4. Template Method Pattern

- **Where It's Used:** The findAll and findByIssuerAndDateRange methods in the service implementations reuse repository methods (e.g., findAll, findByIssuerOrderByDateDesc) while applying custom transformations like DTO mapping.
- **Why We Used It:** This avoids code duplication and ensures that the core repository behavior remains consistent while allowing us to customize output formats.

Summary

In this project, we integrated key design patterns to demonstrate best practices in software engineering. Patterns like Singleton, Adapter, MVC, and Template Method were chosen to address challenges such as database interaction, clean API responses, and separation of concerns. By applying these patterns, the application became more modular, testable, and easier to extend, which is crucial for building scalable systems.