Документација: Рефакторирање на Модели и Услуги

Преглед

Целта на рефакторирањето беше да се стандардизира и поедностави структурата на кодот, додека се решаваат проблемите поврзани со генерирањето на конструктори и се намалува редундантноста. Рефакторирањето вклучуваше промени во моделите и апстракцијата на заеднички својства во базна класа. Дополнително, заедничките service интерфејси беа поедноставени, што ја елиминираше потребата од повторување на методите на секоја специфична индикаторска класа (Дневна, Месечна, Неделна).

---

1. Рефакторирање на Моделите

Пред Рефакторирање:

- Моделите како IndicatorsDaily, IndicatorsWeekly и IndicatorMonthly беа дефинирани секој поединечно со слични полиња (како SMA_50, EMA_50, итн.).
- Секој од овие класи имаше свој сет на конструктори, што доведуваше до редундантен код низ класите.

Стратегија за Рефакторирање:

1. **Создавање на Заедничка Базна Класа (IndicatorsBase)**:
    - Класата IndicatorsBase беше воведена како апстрактна база која ги држи заедничките својства како:
        - id
        - highPrice, lowPrice и други технички индикатори (SMA_50, EMA_50, итн.)
        - signal и issuer.
    - Оваа базна класа е означена како @MappedSuperclass, што значи дека не е директно мапирана на табела во базата на податоци, но нејзините полиња се наследуваат од децата.
2. **Децата на Моделите ја Наследуваат IndicatorsBase**:
    - Моделите IndicatorsDaily, IndicatorsWeekly и IndicatorMonthly сега ја наследуваат класата IndicatorsBase, елиминирајќи ја редунданцијата во полињата.

- Секој од овие модели е означен со анотациите @Entity и @Table за да бидат правилно мапирани на соодветни табели во базата на податоци.

3. **Решавање на Конфликти со Конструктори**:
    - Конфликтот со конструкторите на Lombok беше решен со тоа што се дозволи генерирање на конструктори од страна на IndicatorsBase.
    - Децата на класите веќе не дефинираат експлицитно конструктор без аргументи, избегнувајќи дупликација.

---

2. Рефакторирање на Сервисите

Пред Рефакторирање:

- Секој сервис (на пр. DailyService, MonthlyService, WeeklyService) имаше идентични методи како getAllIndicators(), getIndicatorsById() и saveIndicator(). Овие методи беа дефинирани и повторувани за секој сервис, што доведуваше до редундантен код.

Стратегија за Рефакторирање:

1. **Апстрактен Сервис (BaseIndicatorService)**:
    - Беше воведена нова апстрактна класа за услуга, BaseIndicatorService, која ја апстрахира заедничката логика и ја елиминира повторувањето на кодот.
    - Методите како getAllIndicators(), getIndicatorsById() и saveIndicator() беа преместени во овој базен сервис.
    - Базниот сервис работи со генерички IndicatorRepository, кој обезбедува CRUD операции за моделите IndicatorsBase.
2. **Специфични Сервисни Имплементации (DailyServiceImpl, MonthlyServiceImpl, итн.)**:
    - Секој специфичен сервис (DailyService, MonthlyService, WeeklyService) сега ја наследува класата BaseIndicatorService и ја наследува заедничката функционалност.
    - Овие специфични сервиси веќе не мора да ги дефинираат стандардните методи, но можат да ги преземат или додаваат сопствени логики, доколку е потребно.

3. **Инјекција на Зависности**:
    - Анотацијата @Autowired од Spring беше користена за инјектирање на зависностите на репозиториумите во класите за услуги.
    - Услугите користат интерфејсите за репозиториуми (DailyRepository, MonthlyRepository, итн.) за интеракција со базата на податоци.

---

3. Рефакторирање на Контролери

Пред Рефакторирање:

- Секој контролер (DailyController, MonthlyController, WeeklyController) имаше дуплирани ендпоинти, кои ракуваа со слична функционалност за секој тип на индикатор.
- Овие контролери имаа свој сет на патеки за добивање, зачувување и бришење индикатори според ID, симбол или сигнал.

Стратегија за Рефакторирање:

1. **Консолидирање на Логиката на Контролери**:
    - Беше воведена поедноставена и генерализирана структура на контролер, каде што логиката за ендпоинтите е слична за сите типови индикатори.
    - Контролерот сега комуницира со рефактираните сервиси (DailyService, MonthlyService, WeeklyService), кои го ракуваат бизнис логиката.
    - Заедничка патека @RequestMapping беше користена, со специфични ендпоинти за секој тип на индикатор.
2. **Користење на Наследување за Заеднички Методологии**:
    - Патеките во контролерот (GET, POST, DELETE) се идентични за сите типови индикатори. Како резултат, методите на контролерот беа напишани генерализирано.
    - Оваа поставка ја подобрува одржливоста и го прави кодот пообновлив како нови типови на индикатори може лесно да се додадат без повторување на логика.

---

4. Предности на Рефакторирањето

- **Намалена Редундантност**: Со преместување на заедничките својства и логика во базни класи, кодот за управување со индикаторите беше значително намален.
- **Подобрената Одржливост**: Воведувањето на базни класи и сервиси го олеснува додавањето нови типови на индикатори без дуплицирање на кодот.
- **Зголемена Читливост**: Почистиот и поедноставен код ја подобрува читливоста и ја намалува веројатноста за грешки поради редундантен код.
- **Скалабилност**: Апликацијата лесно може да смести дополнителни типови на индикатори (на пример, неделни, месечни) или други модификации на базната логика со минимални промени на постоечкиот код.
- **Ефикасност**: Со користење на анотации од Lombok и намалување на кодот кој се повторува, времето на развој се скратува, а кодот станува поефикасен за работа.

5. Дизајн Патерн: Шаблон на Стратегија (Strategy Pattern)

Во текот на рефакторирањето, беше применет **Шаблон на Стратегија (Strategy Pattern)** за да се апстрахира заедничката логика за индикаторите. Овој патерн ни дозволува да ја изолираме логиката за CRUD операции во базен сервис (BaseIndicatorService) и да ја оставиме специфичната логика за секој тип индикатор (дневен, неделен, месечен) да се имплементира во неговиот специфичен сервис.