Анализа на архитектурата и дизајн-шаблоните

1. Архитектонски шаблони

- Архитектура на микросервиси:
    - Апликацијата е поделена на повеќе сервиси:
        - main_app за примарната функционалност.
        - prediction_service за предиктивна анализа.
        - strategy_service за анализаторски стратегии.
    - Комуникацијата помеѓу сервисите се реализира преку HTTP API.
    - Docker и docker-compose се користат за контејнеризација и управување со овие сервиси, овозможувајќи независно скалирање, распоредување и изолација на грешки.
- Слоевита архитектура (внатре во секој сервис):
    - Секој сервис следи слоевита структура:
        - Контролер слој: Управува со рутирањето и корисничкиот влез (main_controller.py).
        - Сервисен слој: Ги инкапсулира бизнис-логиките (пр. analysis_service.py, prediction_service.py).
        - Модел слој: Управува со пристапот до податоци и нивната трансформација (stock_model.py).

2. Шаблони за дизајн

- Model-View-Controller (MVC):
    - Папките во main_app:
        - Модели: stock_model.py ги инкапсулира логиките за пристап до базата на податоци преку SQL.
        - Прегледи: HTML шаблони во templates/ се користат за рендерирање на корисничкиот интерфејс.
        - Контролери: main_controller.py управува со корисничкиот влез и координира одговори.

- Repository Pattern::
    - stock_model.py ги апстрахира операциите со базата на податоци, овозможувајќи методи за пребарување, филтрирање и трансформирање податоци за повторна употреба без директно изложување на SQL логиката.
- Strategy pattern:
    - Користен во services/analysis_strategies.py за динамично селектирање на технички индикатори и генерирање сигнали за купување/продавање.
    - Инкапсулацијата на различни алгоритми за анализа овозможува флексибилност и одржливост.

3. Инфраструктура и распоредување

- Docker:
    - Секој сервис има свој Dockerfile, следејќи ги најдобрите практики:
        - Лесна python:slim слика за намалување на големината.
        - Корисници со ограничени привилегии за подобрена безбедност.
        - Изолација на зависности преку requirements.txt.
- Docker-Compose:
    - docker-compose.yaml оркестрира распоредување на сервисите, со дефинирање на зависности помеѓу нив (depends_on).

4. Напредни функционалности

- Предиктивно моделирање (LSTM):
    - prediction_service.py користи:
        - LSTM мрежи за анализа на временски серии.
        - scikit-learn за предобработка на податоци.
        - Keras за креирање и тренирање на длабоки модели.
    - Ова демонстрира Builder шаблон преку конструкција на комплексни LSTM модели на модуларен начин.
- Техничка анализа:
    - analysis_service.py интегрира:
        - Подвижни проседи (Moving Averages), RSI и MACD за анализа на пазарот.
        - Python библиотеката ta го поедноставува имплементирањето.

Шаблон	Каде е користен	Зошто е користен
Микросервиси	Целата апликација	Скалирање, изолација на грешки, независно ажурирање
MVC	main_app	Одвојување на одговорностите, одржливост
Репозиториум	stock_model.py	Декуплирање на логиката за пристап до податоци
Стратегија	analysis_strategies.py	Флексибилност за додавање нови алгоритми
Builder	prediction_service.py	Модуларно градење на комплексни модели

Оваа дизајн-структура овозможува скалабилност, одржливост и флексибилност, во согласност со модерните практики за развој на софтвер. Избраните шаблони, MVC и Strategy Pattern го овозможуваат тоа, како и лесно управување и модуларност. Овие пристапи се идеални за апликации со сложена логика како што е оваа, каде предвидувањата и анализите се изведуваат врз големи количини податоци, во овој случај при анализа на страната на Македонска Берза за секоја компанија за секој ден во последните 10 години.