Анализа на архитектура и дизајн шаблони

1. Архитектонски шеми

Микросервиси

- Ова е дизајн каде апликацијата е поделена на мали, независни сервиси. Секој сервис има специфична улога и може да се стартува и надгради независно.
- **Како го искористивме ние:**
    - Проектот е поделен на неколку сервиси:
        - **Главен сервис (Dians)**: Управува со корисничкиот интерфејс и координацијата.
        - **Prediction Service (Сервис за предвидување)**: Предвидува цени на акции со машинско учење.
        - **Strategy Service (Сервис за стратегии)**: Имплементира стратегии за техничка анализа на акции.
    - Секој сервис има свој app.py, Dockerfile и API за комуникација.
        - Пример: Во main_controller.py, сервисот за стратегии се повикува преку http://strategy_service:5003/analyze.
- **Зошто го користевме:**
    - **Скалабилност:** Секој сервис може независно да се надградува или проширува.
    - **Изолација на грешки:** Грешка во еден сервис (на пример, Prediction Service) нема да влијае на останатите.
    - **Флексибилност:** Лесно додавање нови сервиси без да се наруши постојната структура.

MVC (Model-View-Controller)

- Ова е шаблон за структурирање на апликациите преку поделба на три слоја:
    - **Модел:** Го управува податочното ниво.
    - **Вид:** Го претставува корисничкиот интерфејс.
    - **Контролер:** Ја содржи логиката за поврзување на моделот и видот.

- **Како го искористивме ние:**
    - **Модел (Model):** models/stock_model.py содржи логика за пристап до базата на податоци.

        - Пример: get_stock_data ја презема листата на издавачи со пагинација.
    - **Вид (View):** HTML шаблони (index.html, analysis.html) го претставуваат корисничкиот интерфејс.
        - Шаблоните користат динамички податоци преку Jinja2 (на пример, променливата stock_data).
    - **Контролер (Controller):** main_controller.py управува со рутите и логиката за поврзување на моделите и видот.
        - Пример: Рутата /analysis презема податоци од get_filtered_data_for_analysis и ги прикажува во analysis.html.
- **Зошто го користевме:**
    - **Модуларност:** Логиката, податоците и интерфејсот се одвоени, што го олеснува одржувањето.
    - **Повторна употреба:** Шаблоните и моделите можат да се користат повеќе пати.

2. Дизајнерски шаблони

Репозиториум (Repository Pattern)

- Оваа шема ја апстрахира базата на податоци и обезбедува унифициран интерфејс за пристап до податоците.
- **Како го искористивме ние:**
    - Функциите во models/stock_model.py, како get_stock_data и get_total_issuers_count, ги обработуваат податоците од базата без да откријат детали за самата база.

  Пример:
        - def get_stock_data(page=1, limit=10, table="stock_data"):
        - conn = sqlite3.connect(DB_NAME)
        - query = f"SELECT * FROM {table} LIMIT {limit} OFFSET {(page - 1) * limit}"
        - cursor.execute(query)
        - return cursor.fetchall()
- **Зошто го користевме:**
    - **Инкапсулација:** Кодот за пристап до базата е изолиран, што го прави полесен за менување.
    - **Флексибилност:** Лесно се менува базата (на пример, од SQLite во PostgreSQL) без промени во останатиот код.

Стратегија (Strategy Pattern)

- Шема што овозможува динамички избор на различни алгоритми или стратегии.
- **Како го искористивме ние:**
    - Во main_controller.py, корисникот може да избере стратегија за анализа преку параметарот strategy.

  Пример:

    - chosen_strategy = request.args.get('strategy', 'full').lower()
    - data_payload = {'issuer_data': df.to_dict(orient='records'), 'strategy': chosen_strategy}
    - response = requests.post('http://strategy_service:5003/analyze', json=data_payload)

    - Оваа шема ни овозможи лесна интеграција на стратегии како RSI, MACD, EMA и други.
- **Зошто го користевме:**
    - **Флексибилност:** Лесно додавање нови стратегии без промена на постојниот код.
    - **Модуларност:** Стратегиите се одвоени од главната апликација.

Декоратор (Decorator Pattern)

- Шема што овозможува додавање функционалност на функции или класи без да се менува нивниот изворен код.
- **Како го искористивме ние:**
    - Flask ги користи декораторите за поврзување на рутите со функции.

  Пример:

    - @main_blueprint.route('/analysis')
    - def analysis():
    - Логика за потребната анализа
- **Зошто го користевме:**
    - **Јасност:** Рутите се јасно дефинирани и поврзани со нивната логика.
    - **Повторна употреба:** Истите функции може да се користат со различни декоратори за различна функционалност.

3. Зошто овие шеми се користат?

- **Скалабилност:** Микросервисите и стратегијата овозможуваат проширување на апликацијата со минимален напор.
- **Одржливост:** MVC и Repository Pattern ја прават апликацијата лесна за одржување и тестирање.
- **Флексибилност:** Лесно додавање нови функции, алгоритми или дизајнерски промени.
- **Корисничко искуство:** HTML шаблоните и Flask рутите обезбедуваат динамичен и лесен за употреба интерфејс.

Архитектонска шема	Што е тоа?	Како го искористивме?	Зошто го користевме?
Микросервиси	Системски дизајн каде апликацијата е поделена на мали, независни сервиси.	Проектот има сервиси како Dians, Prediction Service, и Strategy Service. Секој сервис има свој app.py, Dockerfile и HTTP API за комуникација. Пример: main_controller.py повикува http://strategy_service:5003/analyze за анализа.	**Скалабилност:** Секој сервис може да се надгради независно. **Флексибилност:** Лесна интеграција и одвоено развивање.
MVC (Модел-Вид-Контролер)	Шема што ја дели апликацијата на Модел (податоци), Вид (интерфејс), Контролер (логика).	- **Модел:** models/stock_model.py управува со податоците и базата. - **Вид:** HTML шаблони (index.html, analysis.html) за корисничкиот интерфејс. - **Контролер:** main_controller.py ги управува рутите. Пример: /analysis ја презема анализа податоци и ги испраќа до шаблоните.	**Модуларност:** Делите логика, податоци и интерфејс. **Повторна употреба:** Моделите и видовите може да се користат повеќе пати.

Дизајн Шаблон	Што е тоа?	Како го искористивме?	Зошто го користевме?
Репозиториум	Шема што ја апстрахира базата на податоци и обезбедува унифициран интерфејс за пристап.	Функции како get_stock_data, get_total_issuers_count и get_issuer_details во stock_model.py управуваат со податоците преку апстрахирани методи. На пример: get_stock_data ја презема листата на издавачи од базата.	**Инкапсулација:** Го одделува пристапот до базата од останатата логика. **Флексибилност:** Лесна замена на базата на податоци (SQLite и сл.).
Стратегија	Дизајнерска шема што овозможува избор на различни алгоритми во текот на извршување.	Во main_controller.py, стратегијата за анализа (RSI, MACD, EMA) се избира динамички преку параметар strategy. Пример: chosen_strategy = request.args.get('strategy', 'full'). Резултатите потоа се обработуваат од Strategy Service.	**Флексибилност:** Лесно додавање нови стратегии без промена на главната логика.
Декоратор	Шема што овозможува додавање ново однесување на функции динамички.	Flask ги користи декораторите како @app.route за врзување на URL со функции. Пример: @main_blueprint.route('/analysis') ја дефинира рутата за анализа.	**Јасност:** Лесно дефинирање на рути. **Модуларност:** Истата функција може да се користи со различни декоратори за различно однесување.