Илементирани дизајн шаблони

Factory Method Pattern

Во оваа апликација е имплементиран **Factory Method** дизајн шаблонот со цел да се централизира и стандардизира креирањето на објекти како што се репозиториуми, сервиси и конекции кон базата на податоци. Наместо објектите да се инстанцираат директно во API рутите или во бизнис логиката, нивното креирање е делегирано на посебни фабрички функции.

Оваа имплементација се наоѓа во модулот dependencies.py.

Проблем без Factory Method

Доколку не се користи Factory Method шаблонот, секоја рута или сервис би морала директно да креира свои инстанци од репозиториуми или конекции кон базата. Ова доведува до:

- Дуплирање на код
- Цврста поврзаност меѓу рутите и конкретните имплементации
- Потешко одржување и проширување на апликацијата

Имплементација на Factory Method

Factory Method шаблонот е имплементиран преку функции кои ја инкапсулираат логиката за креирање објекти.

```
def get_prices_repo() -> PricesRepository:
    return PricesRepository(conn_factory=get_conn)

def get_symbols_repo() -> SymbolsRepository:
    return SymbolsRepository(conn_factory=get_conn)

def get_technical_service() -> TechnicalAnalysisService:
    return TechnicalAnalysisService(prices_repo=get_prices_repo())
```

Секоја од овие функции претставува **Factory Method**, односно: креира конкретен објект, ги иницијализира неговите зависности, враќа целосно подготвена инстанца.

Factory Method за конекција кон база

Креирањето на конекцијата кон базата е централизирано во connection.py:

```
def get_conn() -> sqlite3.Connection:
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    return conn
```

Оваа функција функционира како фабрика за конекции и обезбедува:

- ❑ Конзистентна конфигурација на сите конекции
- ❑ Централизирана контрола
- ❑ Лесна измена без влијание врз клиентскиот код

Придобивки од Factory Method

- ❑ **Слаба поврзаност** – клиентскиот код не зависи од конкретни имплементации
- ❑ **Подобра одржливост** – измените се прават на едно место
- ❑ **Повторна употреба** – фабриките се користат низ целата апликација
- ❑ **Скалабилност** – лесно додавање нови сервиси или репозиториуми

MVC (Model–View–Controller) Pattern

Архитектурата на backend апликацијата го следи **MVC (Model–View–Controller)** дизајн шаблонот, кој овозможува јасна поделба на одговорностите. Овој пристап ја подобрува читливоста, одржливоста и проширливоста на системот.

1. Model слој

Model слојот е задолжен за работа со податоци и бизнис логика. Во апликацијата, овој слој е имплементиран преку репозиториум класи како:

- ❑ PricesRepository
- ❑ SymbolsRepository

Овие класи: директно комуницираат со базата на податоци, ја инкапсулираат SQL логиката, обезбедуваат доменски методи за пристап до податоци.

2. Controller слој

Controller слојот е имплементиран преку FastAPI рутите:

- routes_prices.py
- routes_symbols.py
- routes_technical.py
- routes_lstm.py

Контролерите: примаат HTTP барања, повикаат соодветни сервиси или репозиториуми, враќаат одговор кон клиентот.Исто така тие **не содржат логика за пристап до база**, со што се одржуваат едноставни и прегледни.

3. View слој

Во REST backend апликација, View слојот е претставен преку:

- JSON одговори
- HTTP статус кодови
- Response модели.

View слојот служи исклучиво за презентација на податоци.

Router како MVC координатор

Фајлот router.py функционира како централен координатор на контролерите:

```
router.include_router(routes_symbols.router)
router.include_router(routes_prices.router)
router.include_router(routes_technical.router)
```

Ова овозможува: модуларна организација на контролерите, јасна структура на API-то и лесно проширување со нови функционалности.

Придобивки од MVC

- **Јасна поделба на одговорности**
- **Подобра одржливост**
- **Лесна надградба и проширување**

- **Подобра прегледност на кодот**

Заклучок

Со имплементацијата на **Factory Method** и **MVC** дизајн шаблоните, апликацијата постигнува чиста, модуларна и скалабилна архитектура. Креирањето на објекти е централизирано и контролирано, додека одговорностите се јасно одделени по слоеви. Овој пристап ја подобрува одржливоста, повторната употреба и целосно се усогласува со дизајн шаблоните опфатени во наставниот материјал.