Анализа на шаблони за дизајн употребени во кодот

Вовед

Во развојот на софтверски системи, употребата на шаблони за дизајн (design patterns) е широко прифатен пристап кој овозможува постигнување на високо ниво на квалитет, флексибилност и одржливост на кодот. Шаблоните за дизајн се проверени решенија за чести проблеми во софтверската архитектура и нивната примена ја намалува сложеноста при развојот, олеснувајќи ја соработката помеѓу тимовите и зголемувајќи ја повторната употребливост на компонентите.

Во овој документ ќе бидат идентификувани и анализирани шаблоните за дизајн кои се применети во нашиот код. Преку детална анализа, ќе се објасни нивната примена, како и придобивките кои произлегуваат од нивната интеграција. Оваа анализа служи како пример за тоа како софтверските архитектонски принципи можат да се применат за развој на ефикасни и скалабилни системи.

Преглед на кодот

Кодот претставува веб апликација за анализа на финансиски податоци која нуди следниве функционалности:

- Читање и обработка на историски финансиски податоци од CSV-датотеки.
- Техничка анализа преку пресметување на индикатори како RSI, EMA, SMA и други.
- Фундаментална анализа базирана на новости за компаниите.
- Предвидување на идни движења на цените со користење на напредни модели за машинско учење како LSTM (Long Short-Term Memory).

Шаблони за дизајн

- ***Singleton***

Singleton е шаблон за дизајн кој осигурува дека одредена класа има само една инстанца во текот на извршувањето на програмата и обезбедува глобална точка за пристап до таа инстанца. Во контекст на софтверските системи, овој шаблон често се користи за

управување со ресурси кои се скапи за создавање, како што се конекции до бази на податоци или логирање.

Причина за примена: Овој шаблон овозможува централизирано управување со конфигурациите и ресурсите на апликацијата, спречувајќи создавање повеќе инстанци кои би можеле да предизвикаат конфликт.

Во нашиот код иако не се користи класичен Singleton pattern со специфична имплементација (како глобален објект), може да се воочи сличната логика. На пример, класа како DataExtractor се користи за собирање на податоци од интернет и се создава само еднаш, по што се користи за да се добијат сите податоци. Инстанцирањето на DataExtractor и неговото користење во целиот процес го следи принципот на едноставно повторно користење на еден објект, кој обезбедува управување со податоците на ефикасен начин.

- ***Strategy pattern***

Стратегијата е дизајн патерн кој овозможува избор на алгоритам или логика во текот на времето, односно во runtime. Овој патерн се користи за дефинирање на различни алгоритми кои можат да се применуваат на исто податочно множество, и му овозможува на клиентскиот код да избере која стратегија ќе ја примени, во зависност од конкретната ситуација. Стратегијата помага да се изолираат специфичните имплементации од основната структура и да се постигне флексибилност, што ги прави процесите поеластични и лесни за модификација.

Примена на овој шаблон во кодот:

Шаблонот е применет за обработка на технички индикатори за анализа на податоци од пазарот на акции. Овој процес вклучува избор и комбинирање на различни алгоритми за пресметување на индикатори како што се SMA (Simple Moving Average) и RSI (Relative Strength Index). Овие индикатори се користат за анализа на историските податоци за акциите, како што се цените и обемот на тргување.

```
data['SMA_10'] = ta.trend.sma_indicator(data['last_price'], window=10)

data['RSI'] = ta.momentum.rsi(data['last_price'], window=14)
```

Во овој дел од кодот, се применуваат два различни технички индикатори:

SMA (Simple Moving Average): Овој индикатор пресметува просечна вредност на цената на акциите за одреден период. Во примерот, window=10 значи дека се користат последните 10 денови од цената за да се пресмета просечната вредност.

RSI (Relative Strength Index): Овој индикатор се користи за мерење на трендови во цената на акцијата, со помош на обемот на растење и опаѓање на цената во одреден период (во примерот, 14 дена).

Причина за примена на Strategy design pattern: Стратегијата е применета во кодот затоа што овозможува висока флексибилност и лесно проширување на системот. Во случај на додавање нови индикатори за анализа, кодот може лесно да се прошири само со додавање нова линија која ќе ја користи истата стратегија, без потреба за значителни промени во основниот алгоритам. На овој начин, новите стратегии за анализа можат да се имплементираат без да се менува структурата на кодот, што е важно кога работиме со динамички податоци кои се менуваат во текот на времето.

- ***Template Design Pattern***

Шаблонскиот дизајн патерн се користи за дефинирање на основната структура на алгоритамот, при што некои чекори од алгоритамот се дефинираат како апстрактни методи кои можат да се имплементираат од подкласи. Овој патерн обезбедува флексибилност за измена на специфични делови од алгоритамот без да се менува целокупната структура.

Во нашиот код, пример за примена на шаблонски патерн е функцијата DataExtractor и HistoricalDataFetcher. Во овие класи, процесот на симнување на податоци од веб-страницата и обработката на тие податоци се дефинирани преку основни чекори, додека пак специфичните имплементации на одредени делови се дефинирани во поткласите.

Пример:

```
class DataExtractor:

  def run(self):

    response = requests.get(self.BASE_URL)

    response.raise_for_status()

    soup = BeautifulSoup(response.text, 'html.parser')

    dropdown_menu = soup.select_one("select#Code")


    company_list = []


    if dropdown_menu:
```

```
    for option in dropdown_menu.find_all("option"):
        code = option.get("value")
        if self.validate_code(code):
            company_list.append(code)

    self.write_to_csv(company_list)
    return company_list
```

Во овој случај, методата run() го дефинира основниот алгоритам за симнување на податоци, додека потклasите како HistoricalDataFetcher го дополнуваат алгоритамот со конкретни детали (на пример, додавање на специфични компании и симнување на историски податоци).

Причина за употреба на шаблонски патерн е што овозможува повторна употреба на истата структура на алгоритмот и измена на специфични делови во зависност од потребите. Ова го прави кодот полесно одржлив и проширлив, бидејќи само делови од алгоритмот се променливи, додека целокупниот тек останува ист.

- ***Factory design pattern***

Фабричкиот дизајн патерн се користи за креирање објекти без да се дефинира конкретна класа на објектот што треба да се создаде. Овој патерн го изолира процесот на креирање на објектот и овозможува на клиентскиот код да работи со апстракцијата на објектите без да мора да се грижи за нивната конкретна имплементација.

Фабричкиот патерн се применува во создавањето на објекти за вчитување и обработка на податоци од веб-страницата. Пример е класата DataExtractor која создава објекти за симнување на податоци за компании од даден URL. На истиот извадок од кодот наведен малку погоре може да се забележи дека фабричката функција е во суштина методата run(), која генерира список на компании (креира објекти) врз основа на HTML содржината што ја добива преку веб-барањето. Методата run() не се грижи за тоа како точно ќе се изврши ова симнување или обработка на податоците (не се грижи за конкретните класи кои ја претставуваат структурата на податоците), туку само ги предава тие работи на специфичните помошни функции како write_to_csv() и validate_code(), кои ја обработуваат логиката на специфичните операции.

Причина за употреба на фабрички патерн е тоа што овозможува лесно креирање на објекти без да се вмешува клиентскиот код во конкретната имплементација на тие објекти. Фабричката функција го олеснува менаџирањето на зависностите и ги скрива деталите за креирање на објектите, што го прави кодот поедноставен за одржување и проширување.

Observer Design Pattern

Обсервер дизајн патернот претставува архитектонска стратегија која овозможува еден објект, познат како субјект, да ја известува групата зависни објекти, наречени обсервери, за секоја промена во неговата состојба. Притоа, субјектот не поседува знаење за тоа кои обсервери постојат или како тие ќе реагираат. Овој пристап е исклучително корисен за дизајнирање на апликации каде што промената во состојбата на еден елемент треба автоматски да иницира специфични активности или ажурирања во други делови од системот, овозможувајќи висока флексибилност, модуларност и независност на компонентите.

Обсервер патерн се имплементира во делот каде што се прави следење на различни показатели (поставување на индикатори) за време на анализа на технички податоци. Еден пример е кога се пресметуваат различни технички индикатори (како што се SMA_10, RSI, и други) и се следат нивните промени во зависност од времето.

Пример од кодот:

Пресметување на технички индикатори

data['SMA_10'] = ta.trend.sma_indicator(data['last_price'], window=10)

data['RSI'] = ta.momentum.rsi(data['last_price'], window=14)

data['MACD'] = ta.trend.macd(data['last_price'])

Овде, секој од овие индикатори може да се третира како "обсервер" на промените во last_price. Како што се менува цената на акцијата (last_price), се пресметуваат нови вредности за индикаторите (обсервери), кои автоматски се ажурираат кога ќе се промени податокот на кој се базираат.

Причината за користење на обсервер патерн во овој контекст е да се овозможи независна реакција на промените во податоците без да се интегрираат сите детали на извршување. Индикаторите се "набљудувачи" кои се ажурираат автоматски секој пат кога податоците за цената ќе се променат, без потреба да се дефинираат конкретни функции за секоја промена.

Придобивките од користењето на овој патерн се видливи во неговата флексибилност и опсег. Системот може да продолжи да работи ефикасно без да се мешаат детали за конкретните индикатори, а новите индикатори или реакции можат лесно да се додаваат или отстрануваат без големи модификации на основниот код. Ваквата архитектура обезбедува лесно проширување и флексибилност на целиот систем.

Заклучок

Извршивме анализа на неколку клучни дизајн шаблони, меѓу кои Singleton, Factory, Template, Strategy, и Observer, со акцент на нивната примена во развојот на софтвер за анализа на финансиски податоци. Преку конкретни примери од кодот, се прикажа како овие шаблони го олеснуваат креирањето на ефикасни, флексибилни и модуларни софтверски решенија.

Секој од разгледаните шаблони има своја специфична улога: Singleton обезбедува единствена точка за управување со ресурси, Factory го изолира процесот на креирање објекти, Template дефинира основна структура на алгоритамите со можност за адаптација, Strategy овозможува избор на алгоритми за анализа, додека Observer овозможува реакција на промени во податоците.

Придобивките од примената на дизајн шаблони се повеќекратни: зголемена скалабилност, намалена сложеност, повторна употребливост на кодот, и подобрена модуларност. Покрај тоа, тие придонесуваат за поедноставна одржливост и лесно прилагодување на системот на нови барања.

Заклучокот од оваа анализа е дека рационалната примена на шаблони за дизајн не само што ја подобрува архитектурата на софтверот, туку и овозможува тимовите да работат поефикасно, зголемувајќи ја продуктивноста и квалитетот на производот. Овие принципи се клучни за изградба на современи софтверски системи кои бараат високо ниво на ефикасност и адаптивност.