**1. Модуларно програмирање**

Проектот е организиран во различни фајлови/модули врз основа на функционалност, што го прави лесен за развој, тестирање и одржување. Секој фајл веројатно обработува специфични одговорности:

* **app.py**:
  + Главен влезен точка на апликацијата. Овде програмата веројатно инициира, оркестрира компоненти и ја извршува основната логика.
* **app\_functions.py**:
  + Содржи повторно користливи функции кои ги поддржуваат app.py или други делови од програмата.
* **config.py**:
  + Управува со конфигурациски поставки, како што се API клучеви, креденцијали за база на податоци или други променливи на животната средина.
* **data\_processor.py**:
  + Обработува задачи поврзани со податоци, како што се чистење, трансформирање или агрегација на податоци за анализа или трговија.
* **scrape.py**:
  + Веројатно главен фајл за оркестрирање на операции за веб-скрапинг.
* **scrape\_functions.py**:
  + Содржи помошни функции специјализирани за веб-скрапинг, како што се повлекување на страници, парсирање на HTML или ракување со грешки.
* **trade\_signals.py**:
  + Одговорен за генерирање или пресметување на трговски сигнали врз основа на индикатори, обработени податоци или однапред дефинирани правила.
* **indicators.py**:
  + Имплементира различни трговски индикатори (на пр. движење на просечни, RSI, MACD) што се користат за донесување одлуки во трговската логика.

**2. Dockerизирана апликација**

Присуството на Dockerfile и docker-compose.yml сугерира дека проектот е контејнеризиран за лесно имплементирање и репродуктивност.

* **Dockerfile**:
  + Описува чекори за изградба на Docker слика за проектот, вклучувајќи инсталирање на зависности, копирање на изворни фајлови и поставување на влезен точка.
* **docker-compose.yml**:
  + Управува со повеќе сервиси (на пример, апликации, бази на податоци или други поддржувачки услуги) и дозволува тие да се стартуваат заедно со еден команд.

**3. Која методологија се следи?**

Овој проект изгледа следи комбинација на:

**1. Модуларно програмирање:**

* Кодот е поделени во самостојни модули на базата на функционалност.
* Поттикнува **разделување на загрижености** и го подобрува **повторната употребливост на кодот**.

**2. Функционално програмирање:**

* Користењето на app\_functions.py, scrape\_functions.py и слични фајлови сугерира преференца за функции над класи за имплементирање на логиката.

**3. Контејнеризација:**

* Користејќи Docker, проектот ги прифаќа модерните практики за имплементација, обезбедувајќи конзистентност преку различни средини.

**4. Дизајн управуван од доменот (DDD) (до некоја мера):**

* Секој модул одговара на домен или карактеристика на апликацијата (на пример, скрапинг, трговски сигнали, обработка на податоци).

**Предности на овој пристап:**

1. **Скалабилност**: Модулите можат да се прошируваат или заменуваат независно без да влијаат на целиот проект.
2. **Одржливост**: Секој модул има јасна одговорност, што го прави поедноставен за дебагирање и ажурирање.
3. **Тестирање**: Поединечните компоненти или функции можат да се тестираат поединечно.
4. **Репродуктивност**: Docker обезбедува конзистентност на средината преку сите имплементации.
5. **Соработка**: Разработувачите можат да работат на различни модули истовремено.

**Кога да се користи оваа методологија?**

Оваа структура е идеална за проекти каде што:

* Постојат различни задачи кои логички можат да се поделат (на пример, скрапинг, обработка, трговија).
* Скалабилноста и одржливоста се клучни фактори.
* Имплементацијата треба да биде конзистентна преку развојни, тестни и продукциски средини.  
    
    
  За да ја стартнете апликацијата мора да ги напишете овие 2 команди во 2 терминала:

*# Terminal 1* flask --app api run --port=5007   
*# Terminal 2* flask --app app run --port=5001