Шаблонот за дизајн на Стратегија се користи за справување со различни пристапи за преземање податоци од берзата од веб-локацијата. Интерфејсот DataFetchingStrategy ја дефинира структурата за методите за преземање податоци (fetch\_codes и fetch\_data\_for\_code), а WebScrapingStrategy обезбедува конкретна имплементација која користи веб-стружење за извлекување податоци. Овој дизајн овозможува идна проширливост - на пример, ако треба да преземеме податоци од API или база на податоци наместо да гребеме, можеме да создадеме нови класи на стратегии кои го спроведуваат истиот интерфејс без да го менуваме постоечкиот код. Со тоа, апликацијата останува флексибилна и приспособлива на различни извори на податоци, осигурувајќи дека логиката за преземање е изолирана и повторно употреблива. Овој модуларен пристап е особено важен овде бидејќи методите за преземање зависат од специфична логика (како HTML парсирање и асинхрони барања), кои може да варираат во зависност од изворот или форматот на податоците.

class DataFetchingStrategy:

def fetch\_codes(self, url):

raise NotImplementedError

def fetch\_data\_for\_code(self, session, url, start\_date, end\_date, code):

raise NotImplementedError

class WebScrapingStrategy(DataFetchingStrategy):

def \_\_init\_\_(self, base\_url):

self.base\_url = base\_url

def fetch\_codes(self, url):

response = requests.get(url)

dropdown\_values = []

if response.status\_code == 200:

tree = HTMLParser(response.text)

dropdown = tree.css\_first('select.form-control.dropdown')

if dropdown:

dropdown\_values = [option.attributes['value'] for option in dropdown.css('option') if 'value' in option.attributes]

return dropdown\_values

async def fetch\_data\_for\_code(self, session, url, start\_date, end\_date, code):

data = {

'FromDate': start\_date,

'ToDate': end\_date,

'Code': code

}

headers = {

'User-Agent': 'Mozilla/5.0',

'Content-Type': 'application/x-www-form-urlencoded',

'Referer': url,

'Origin': 'https://www.mse.mk',

}

async with session.post(url, data=data, headers=headers) as response:

if response.status == 200:

page = await response.text()

tree = HTMLParser(page)

rows = tree.css('table tr')

data = []

for row in rows:

cells = row.css('td')

if len(cells) > 8:

data.append({

"code": code,

"date": convert\_date\_format(cells[0].text()),

"last\_transaction\_price": safe\_float(cells[1].text()),

"max\_price": safe\_float(cells[2].text()),

"min\_price": safe\_float(cells[3].text()),

"avg\_price": safe\_float(cells[4].text()),

"percent\_change": safe\_float(cells[5].text().replace(',', '.')),

"quantity": safe\_float(cells[6].text()),

"best\_traded": safe\_float(cells[7].text()),

"total\_traded": safe\_float(cells[8].text()),

})

return data

return []