ДОКУМЕНТАЦИЈА ЗА ИМПЛЕМЕНТИРАНИ ШАБЛОНИ (DESIGN PATTERNS)

1. ШАБЛОН: MVC (MODEL-VIEW-CONTROLLER)

Кодот на апликацијата имплементира **MVC архитектонски шаблон**, кој е основен за Django апликациите.

- **Model:** Работи со податоците и ги дефинира моделите.
- **View:** Логиката е дефинирана во класите како што се StartingPageView, SearchResultsViewTehnicka, и други.
- **Controller:** Django автоматски ги поврзува барањата со соодветните Views преку URL конфигурацијата.

Пример:

Дел од View логика:

```
class StartingPageView(View):
    def get(self, request):
        mse_csv_path = os.path.join(settings.BASE_DIR, "za_homepage", "mse_table.csv")
        sei_net_csv_path = os.path.join(settings.BASE_DIR, "za_homepage",
"sei_net_news_only.csv")

        mse_table_data = []
        with open(mse_csv_path, newline='', encoding="utf-8-sig") as csvfile:
            reader = csv.DictReader(csvfile)
            for row in reader:
                try:
                    change = float(row["% пром."].replace(",", "."))
                    if change > 0:
                        row["color"] = "#2DCE89"  # Green
                    elif change < 0:
                        row["color"] = "#F5365C"  # Red
                    else:
                        row["color"] = "#5E5DFF"  # Blue
                except ValueError:
                    row["color"] = "#000000"  # Black
                mse_table_data.append(row)

        context = {
            "table_data": mse_table_data,
        }
        return render(request, "app/starting_page.html", context)
```

Во овој дел:

- **View:** Логиката е имплементирана во `StartingPageView` класа. Таа обработува податоци од CSV датотека и создава контекст за рендерирање.
- **Template (View во MVC):** Рендерирањето на податоците се врши преку `starting_page.html`.

2. ШАБЛОН: TEMPLATE METHOD

Класите како SearchResultsViewTehnicka наследуваат од TemplateView на Django. Ова е пример за **Template Method** шаблонот каде основната структура на алгоритмот е дефинирана во родителската класа (TemplateView), додека специфичната имплементација е обезбедена преку get_context_data методот.

Пример:

```
class SearchResultsViewTehnicka(LoginRequiredMixin, TemplateView):
    template_name = "app/search_result_page-tehnicka.html"

    def get_context_data(self, **kwargs):
        query = self.request.GET.get("query", "").upper()
        context = super().get_context_data(**kwargs)
        context["query"] = query
        context["timeframe"] = self.request.GET.get("timeframe", "daily")
        # Логиката за обработка на податоци...
        return context
```

Во овој дел:

- `TemplateView` служи како основа за рендерирање на HTML шаблони.
- Методот `get_context_data` е специјално прилагоден за да овозможи обработка и додавање на податоци релевантни за техничката анализа, пред да бидат испратени до HTML шаблонот. Ова овозможува динамично вчитување на информации и визуелизации што се прилагодени на барањето на корисникот.

Функциите `kod_za_generiranje_dijagram` и `kod_za_generiranje_tabela` следат шаблон за обработка на податоци од CSV:

- **Чекор 1:** Вчитување на датотеката.
- **Чекор 2:** Чистење и форматирање на податоците.
- **Чекор 3:** Претворање на податоците во соодветен формат за излез (JSON за API одговори).

Пример:

```
def kod_za_generiranje_dijagram(filename, kolku_unazad_vo_denovi, atribbute):
    kolku_unazad_vo_denovi = int(kolku_unazad_vo_denovi)

    # Вчитување на податоци
    data_file_path = os.path.join(settings.BASE_DIR, 'Data', filename)
    df = pd.read_csv(data_file_path)
    df = df.head(kolku_unazad_vo_denovi)

    # Чистење на податоци
    df['Датум'] = pd.to_datetime(df['Датум'], format='%d.%m.%Y')
```

```
    df[atribbute] = df[atribbute].str.replace('.', '').str.replace(',',
'.').astype(float)

    # Генерирање на излез
    labels = df['Датум'].dt.strftime('%d %b %Y').tolist()
    data = df[atribbute].tolist()
    response_data = {
        "labels": labels,
        "datasets": [{
            "label": atribbute,
            "data": data,
            "borderColor": "#5E72E4",
            "borderWidth": 2,
            "fill": False,
        }]
    }
    return JsonResponse(response_data)
```

Во овој дел предности од Template Method Design Pattern:

- Ги раздвојува чекорите за обработка.
- Лесно прилагодување на секој чекор без промена на останатите.

3. ШАБЛОН: STRATEGY

Овој шаблон се користи за да се имплементираат различни стратегии за обработка на податоци, како што се филтрирање на податоци според временски рамки (`daily`, `weekly`, `monthly`). Методот `_filter_data` имплементира различни стратегии за обработка на податоците.

Пример:

```
if timeframe == "daily":
    filtered_data = df[df["Датум"] >= df["Датум"].max() -
pd.Timedelta(days=30)]
elif timeframe == "weekly":
    filtered_data = df[df["Датум"] >= df["Датум"].max() -
pd.Timedelta(weeks=12)]
elif timeframe == "monthly":
    filtered_data = df[df["Датум"] >= df["Датум"].max() -
pd.Timedelta(days=365)]
else:
    context["error"] = "Невалидна временска рамка е избрана."
```

Во овој дел:

- **Различни стратегии:** `daily`, `weekly`, `monthly`.
- Секој избор имплементира различна стратегија за филтрирање.

4. ШАБЛОН: FACTORY PATTERN ЗА SELENIUM WEBDRIVER

Factory Pattern се користи за конфигурација и иницијализација на **WebDriver** за различни сценарија, како на пример поставување на прилагодени опции за преземање на датотеки.

Пример:

```
def setup_driver(download_dir):
    """Sets up the Selenium WebDriver with a custom download directory."""
    chrome_options = Options()
    prefs = {
        "download.default_directory": download_dir,
        "download.prompt_for_download": False,
        "download.directory_upgrade": True,
        "plugins.always_open_pdf_externally": True,
    }
    chrome_options.add_experimental_option("prefs", prefs)
    driver = webdriver.Chrome(options=chrome_options)
    return driver
```

Во овој дел Factory pattern:

- Ја апстрахира конфигурацијата на WebDriver.
- Го прави кодот повторно употреблив за различни случаи на употреба.

Зошто се користат овие шаблони?

1. **MVC:** Обезбедува јасна поделба на одговорностите меѓу слоевите, што го прави кодот поорганизиран и полесен за одржување.
2. **Template Method:** Овозможува повторна употреба на базичната логика и лесно прилагодување за специфични случаи.
3. **Strategy:** Овозможува флексибилност при додавање нови стратегии за обработка на податоците без да се менува основната структура на апликацијата.
4. **Factory Pattern за Selenium:** Ја поедноставува иницијализацијата на WebDriver со специфични конфигурации.