Model-View-Controller (MVC)

Шаблонот Model-View-Controller (MVC) е софтверски архитектонски шаблон кој се користи за организирање на кодот со одвојување на управувањето со податоци, корисничкиот интерфејс и логиката на апликацијата. Ова одвојување го прави кодот полесен за разбирање, одржување и скалирање.

Компоненти на MVC

1. **Model (Модел)** ○ Моделот е одговорен за управување со податоците на апликацијата, деловната логика и правилата. Тој директно комуницира со базата на податоци или други извори на податоци и обезбедува интерфејс за Контролерот за пристап или манипулирање со податоците.
    - Примери за одговорности:
        - Преземање или ажурирање на цените на акции од база на податоци.
        - Валидација на внесените податоци пред зачувување.
2. **View (Изглед)** ○ Изгледот е одговорен за прикажување на податоците на корисникот. Тој го ракува корисничкиот интерфејс (UI) и ги прикажува информациите од Моделот на разбирлив начин.
    - Примери за одговорности:
        - Прикажување на тековните цени на акции на веб-страница.
        - Покажување пораки за грешка ако податоците не се валидни.
3. **Controller (Контролер)** ○ Контролерот делува како посредник помеѓу Моделот и Изгледот. Тој го обработува корисничкиот внес, повикува методи на Моделот и одредува кој Изглед треба да ги прикаже податоците.
    - Примери за одговорности:
        - Обработка на барање на корисникот за филтрирање на акции според одреден критериум.
        - Ажурирање на базата на податоци кога корисникот додава нова акција во својата листа.

Придобивки од користење на MVC

1. **Одвојување на одговорности** ○ MVC ја дели логиката на апликацијата на три посебни слоеви, што го прави кодот полесен за разбирање и менување. Развивачите можат да работат на Моделот, Изгледот или Контролерот независно без да влијаат на другите компоненти.
2. **Подобрена одржливост** ○ Промените во деловната логика или базата на податоци (Моделот) не влијаат на корисничкиот интерфејс (Изгледот) и обратно. Ова

одвојување го прави отстранувањето на грешки и додавањето на нови функции едноставно.

3. **Повторна употребливост** ○ Компонентите во MVC шаблонот се модуларни, овозможувајќи повторна употреба на кодот во различни делови од апликацијата или дури и во повеќе проекти.
4. **Скалибилност** ○ Јасната структура на MVC го олеснува скалирањето на апликациите кога се додаваат нови функции или модули.

Зошто MVC за апликацијата?

1. **Јасност**: Со одвојување на кодот во Модели, Изгледи и Контролери, структурата на апликацијата ќе биде полесна за разбирање и навигација.
2. **Флексибилност**: Со АПИ/микросервиси кои управуваат со некои функционалности, шаблонот MVC осигурува дека апликацијата останува модуларна и приспособлива за идни промени или надградби.
3. **Подготовка за контејнеризација**: Јасното одвојување на одговорностите го олеснува контејнеризирањето на специфични функционалности и нивното независно распоредување во облакот.
4. **Конзистентност**: Шаблонот MVC воспоставува конзистентни практики низ апликацијата, намалувајќи го дуплирањето на кодот и правејќи ја кодната база почиста и професионална.

Како ние конкретно го имплементиравме овој шаблон?

Нашата имплементација е од т.н. „повеќеслоен MVC“ кој е стандард при изработка на веб апликации во Spring рамката. Имено, помеѓу моделот и контролерот постојат два слоја. Првиот слој е т.н. репозиториум и тука се чува минимално количество на хардкодирани податоци кои се неопходни за креирање на објекти од типот модел. Репозиториумите се употребени од т.н. сервиси кои всушност се интерфејси кои дефинираат имплементација за имплементацискиот подслој на сервисниот слој. Овие имплементации дефинираат методи кои ќе ги користат методите во контролерот а користат податоци од репозиториумот. Со додавање на овие два слоја се намалува зависноста меѓу моделот и контролерот, но во исто време моделот се создава хиерархија: „контролер-сервис-репозиториум-модел“ при што контролерот е на највисоко ниво а моделот на најниско. Со ова, промени во логиката на пониско ниво нема влијание на логиката на повисоко ниво. Сето ова е реализирано во Java код, додека пак view-та, тука познати и како темплејти се реализирани со html и css код, но со користење на thymeleaf engine кој ги прикажува динамичките елементи на темплејтот.

Заклучок

Со имплементација на шаблонот за дизајн MVC, апликацијата ќе стане поорганизирана, полесна за одржување и подготвена за идни развојни предизвици. Овој пристап е усогласен со целите на четвртата домашна работа, осигурувајќи робусна структура за интегрирање на АПИ, микросервиси и контејнеризирано распоредување.