**Дизајн шаблони имплементирани во нашиот проект**

Во нашиот проект ги имаме имплементирано следните дизајн шаблони кои помагаат за подобра организација на кодот, лесна одржливост и зголемена повторна употребливост на компонентите. Овие шаблони ни овозможуваат кодот да биде јасен, структуриран и лесно прошиирлив.

# **Singleton Pattern (Шаблон за синглтон)**

Во нашиот проект користиме Singleton Pattern во сервисните класи, како што е IssuerService. Овој шаблон ни гарантира дека ќе имаме само една инстанца од оваа класа во рамките на Spring контекстот.

Пример од кодот:

@Service

public class IssuerService {

сервисот е автоматски инјектиран со @Autowired

}

Со употреба на @Service аннотацијата, Spring автоматски создава само една инстанца на IssuerService, што ни овозможува поефикасно користење на ресурсите и избегнување на непотребни објекти во меморијата.

# **2. Factory Pattern (Шаблон за фабрика)**

Factory Pattern го користиме кога сакаме да создадеме објекти без директно да користиме оператор new. Ова ни овозможува флексибилност при менување на начинот на создавање на објектите без да влијаеме на кодот кој ги користи.

Пример од кодот:

public List<CsvData> getDataForIssuer(String issuer) {

List<CsvData> filteredData = new ArrayList<>();

// Креираме нов објект на CsvData без директно да го иницираме

filteredData.add(new CsvData(

data[0], data[1], data[2], data[3], data[4],

data[5], data[6], data[7], data[8], data[9]

));

return filteredData;

}

Оваа имплементација ни овозможува лесно да ја менуваме структурата на создадените објекти без да го менуваме остатокот од кодот.

# **3. Observer Pattern (Шаблон за набљудувач)**

**Observer Pattern** се користи за следење и реакција на промени во податоците. Во нашиот проект, клиентската страна ја следи состојбата на серверот и автоматски ги обновува податоците кога се случува промена.

Пример од кодот:

// AJAX повик за обновување на графикот

document.getElementById('issuerForm').addEventListener('submit', function(event) {

var xhr = new XMLHttpRequest();

xhr.open('POST', '/filter', true);

xhr.onreadystatechange = function () {

if (xhr.readyState == 4 && xhr.status == 200) {

// Обновување на графикот

}

};

xhr.send("issuer=" + issuer);

});

Овој механизам овозможува клиентската страна автоматски да ги обновува податоците во зависност од одговорот на серверот, што е карактеристично за **Observer Pattern**.

# **4. MVC Pattern (Модел-Вид-Контролер)**

Нашата апликација ја следи **MVC архитектурата**, што значи дека логиката, податоците и прикажувањето се одделени во три различни слоја:

* **Model** (Модел) – ги содржи податоците и логиката за пристап до нив.
* **View** (Вид) – го претставува корисничкиот интерфејс.
* **Controller** (Контролер) – управува со барањата и ја поврзува логиката со прикажувањето.

Пример од кодот:

@Controller

public class IssuerController {

@GetMapping("/")

public String getAllIssuers(Model model) {

List<String> issuerNames = issuerService.getAllIssuers();

model.addAttribute("issuers", issuerNames);

return "issuerData";

}

Оваа организација ни овозможува лесна модификација на секој дел од апликацијата без да влијаеме на другите компоненти.

# **5. Template Method Pattern (Шаблон за метод)**

Користиме **Template Method Pattern** кога имаме методи со слична структура, но со мали разлики во нивната имплементација. Ова ни овозможува да дефинираме основни чекори во еден метод, додека специфичните имплементации ги дефинираме во посебни методи.

Пример од кодот:

public List<String> getAllDates() {

return readCsvColumn(0);

}

public List<Double> getAllClosingPrices() {

return readCsvColumn(4);

}

private <T> List<T> readCsvColumn(int columnIndex) {

List<T> values = new ArrayList<>();

// Логика за читање на податоци од CSV

return values;

}

Со оваа имплементација, избегнуваме дуплирање на кодот и ги организираме методите во структуриран начин.

## **СУМАРИЗАЦИЈА:**

Во нашиот проект ги применуваме неколку дизајн шаблони кои придонесуваат за неговата ефикасност, одржливост и проширливост:

1. **Singleton Pattern** – За сервисни класи кои треба да имаат само една инстанца.
2. **Factory Pattern** – За креирање објекти на унифициран начин.
3. **Observer Pattern** – За автоматско ажурирање на клиентската страна при промени во серверот.
4. **MVC Pattern** – За јасно раздвојување на моделот, контролерот и погледот.
5. **Template Method Pattern** – За намалување на дупликација и подобрување на организацијата на кодот.

Овие шаблони ни овозможуваат нашиот код да биде поразбирлив, полесен за одржување и подготвен за идни надградби.