Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №4

Технології розроблення програмного забезпечення

Шаблони «singleton», «iterator», «proxy», «state», «strategy»

Варіант 30

Виконав
студент групи ІА-13
Якін С. О.

Перевірив:
Мягкий М.Ю.

Київ 2023

Тема: Шаблони «singleton», «iterator», «proxy», «state», «strategy»

Завдання:

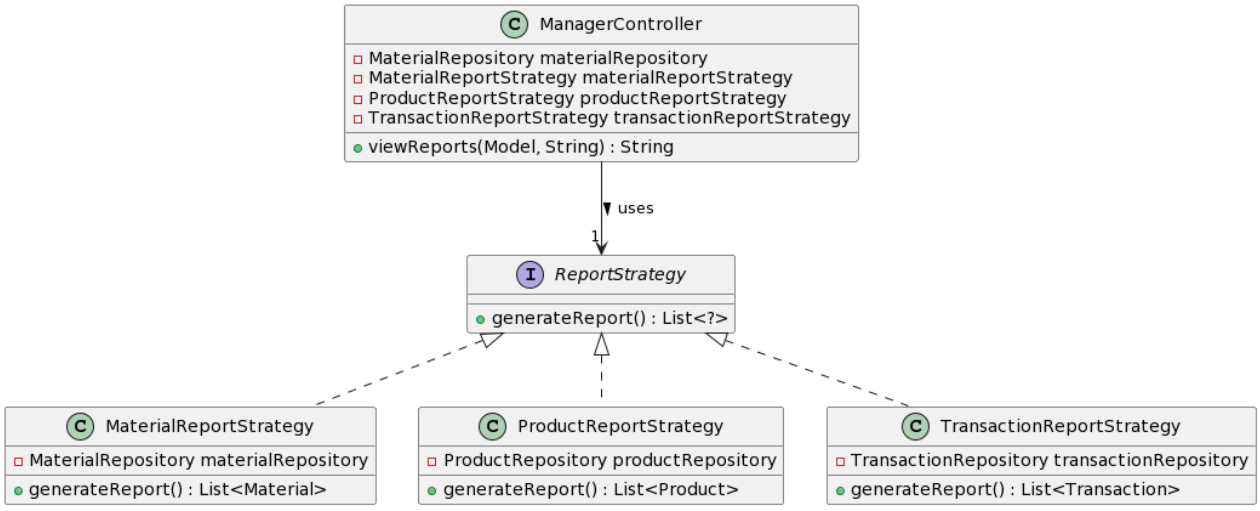
1. Ознайомитися з короткими теоретичними відомостями.
2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3. Застосування одного з розглянутих шаблонів при реалізації програми.

Хід роботи:

Тема:

Система складського обліку виробництва.

Згідно мого завдання, було виконано шаблон проектування «Strategy». У якості реалізації, вирішив використовувати його для функціонального перегляду звітів для користувача (Начальника складу).



Як ми бачимо, в даному випадку, ManagerController є контекстом, який використовує стратегії. Він володіє посиланням на інтерфейс ReportStrategy і може використовувати його різні реалізації.

ReportStrategy є інтерфейсом стратегії, який визначає метод generateReport. Цей інтерфейс є основним для різних конкретних стратегій.

MaterialReportStrategy, ProductReportStrategy, і TransactionReportStrategy є конкретними реалізаціями інтерфейсу стратегії. Кожна з цих класів реалізує метод generateReport відповідно до своєї доменної логіки.

Шаблон проектування "Strategy" дозволяє змінювати поведінку об'єкта за допомогою делегування деяких операцій на різні об'єкти-стратегії. У нашому випадку, ManagerController може змінювати спосіб генерації звіту, обираючи відповідну стратегію (MaterialReportStrategy, ProductReportStrategy, або TransactionReportStrategy). Це дозволяє легко розширювати або змінювати логіку генерації звітів без необхідності змінювати код контролера.

Цей шаблон особливо корисний у ситуаціях, де потрібна гнучкість вибору між декількома варіантами поведінки або алгоритмами. Він також сприяє слабкому зв'язку між класами, оскільки контекст не залежить від конкретних реалізацій стратегій, а лише від їх інтерфейсів.

Код паттерну:

ReportStrategy:

```java
package com.example.warehouse.services.strategies;

import java.util.List;

9 usages  3 implementations
public interface ReportStrategy {
    1 usage  3 implementations
    List<?> generateReport();
}
```

MaterialReportStrategy:

```java
package com.example.warehouse.services.strategies.reports;
import com.example.warehouse.entity.Material;
import com.example.warehouse.repositories.MaterialRepository;
import com.example.warehouse.services.strategies.ReportStrategy;
import lombok.AllArgsConstructor;
import org.springframework.data.domain.Sort;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
@AllArgsConstructor
public class MaterialReportStrategy implements ReportStrategy {
    private final MaterialRepository materialRepository;

    1 usage
    @Override
    public List<Material> generateReport() {

        return materialRepository.findAll(Sort.by( ...properties: "id").ascending());
    }
}
```

ProductReportStrategy:

```java
package com.example.warehouse.services.strategies.reports;
import ...

@Service
@AllArgsConstructor
public class ProductReportStrategy implements ReportStrategy {
    private final ProductRepository productRepository;
    1 usage
    @Override
    public List<Product> generateReport() {

        return productRepository.findAll();
    }
}
```

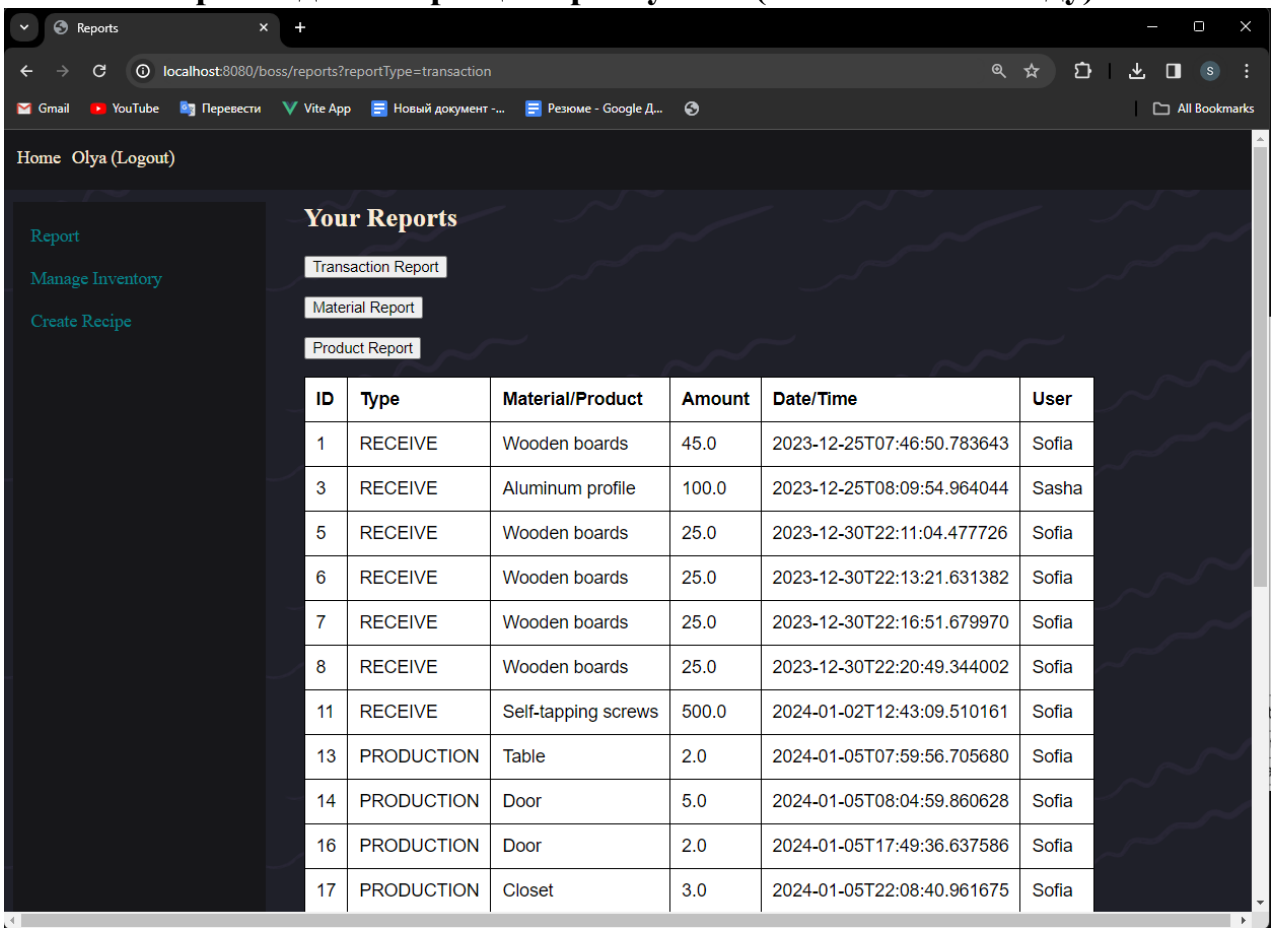
TransactionReportStrategy:

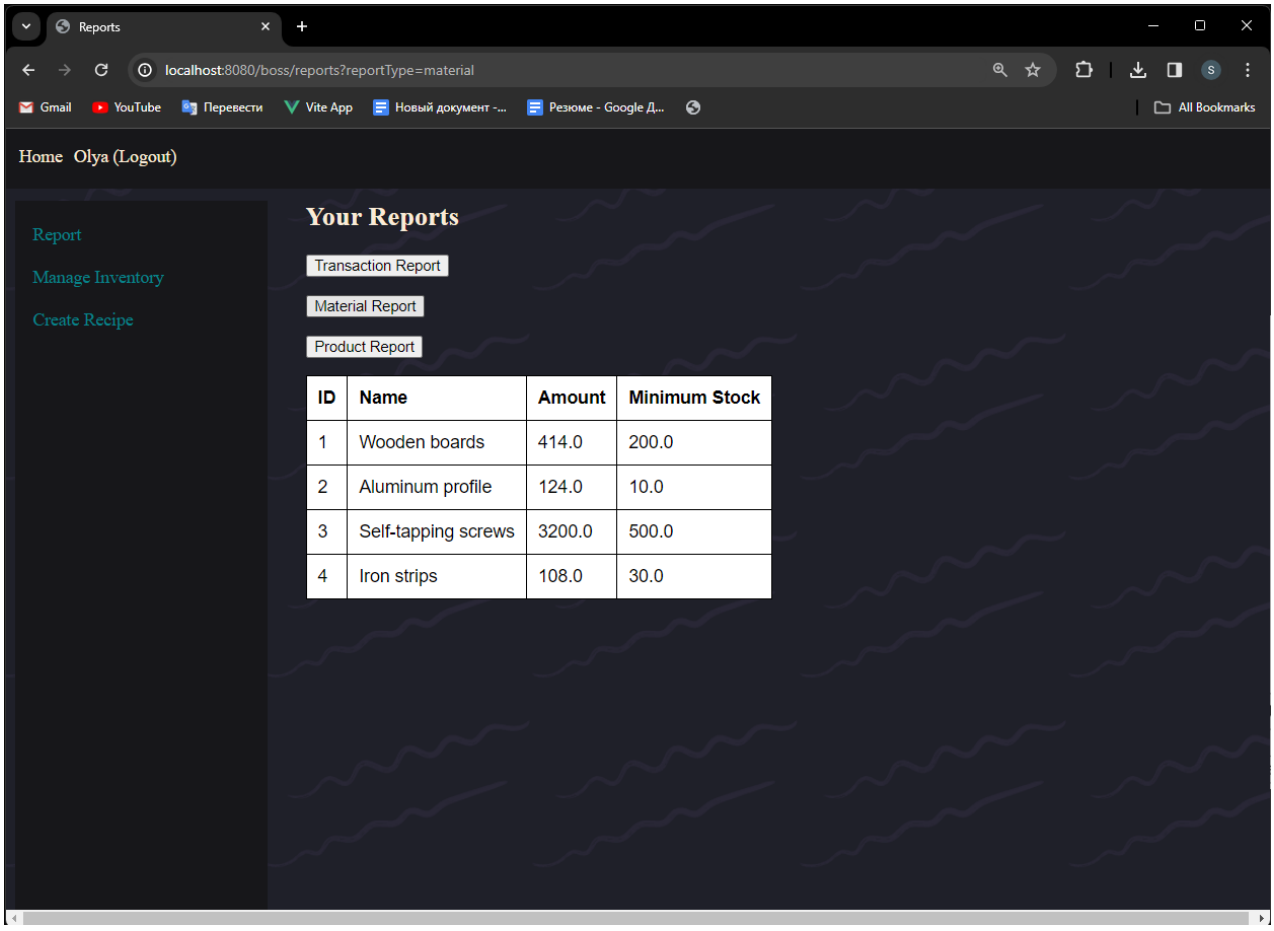
```java
package com.example.warehouse.services.strategies.reports;

import com.example.warehouse.entity.Transaction;
import com.example.warehouse.repositories.TransactionRepository;
import com.example.warehouse.services.strategies.ReportStrategy;
import lombok.AllArgsConstructor;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
@AllArgsConstructor
public class TransactionReportStrategy implements ReportStrategy {
    private final TransactionRepository transactionRepository;

    1 usage
    @Override
    public List<Transaction> generateReport() { return transactionRepository.findAll(); }
}
```

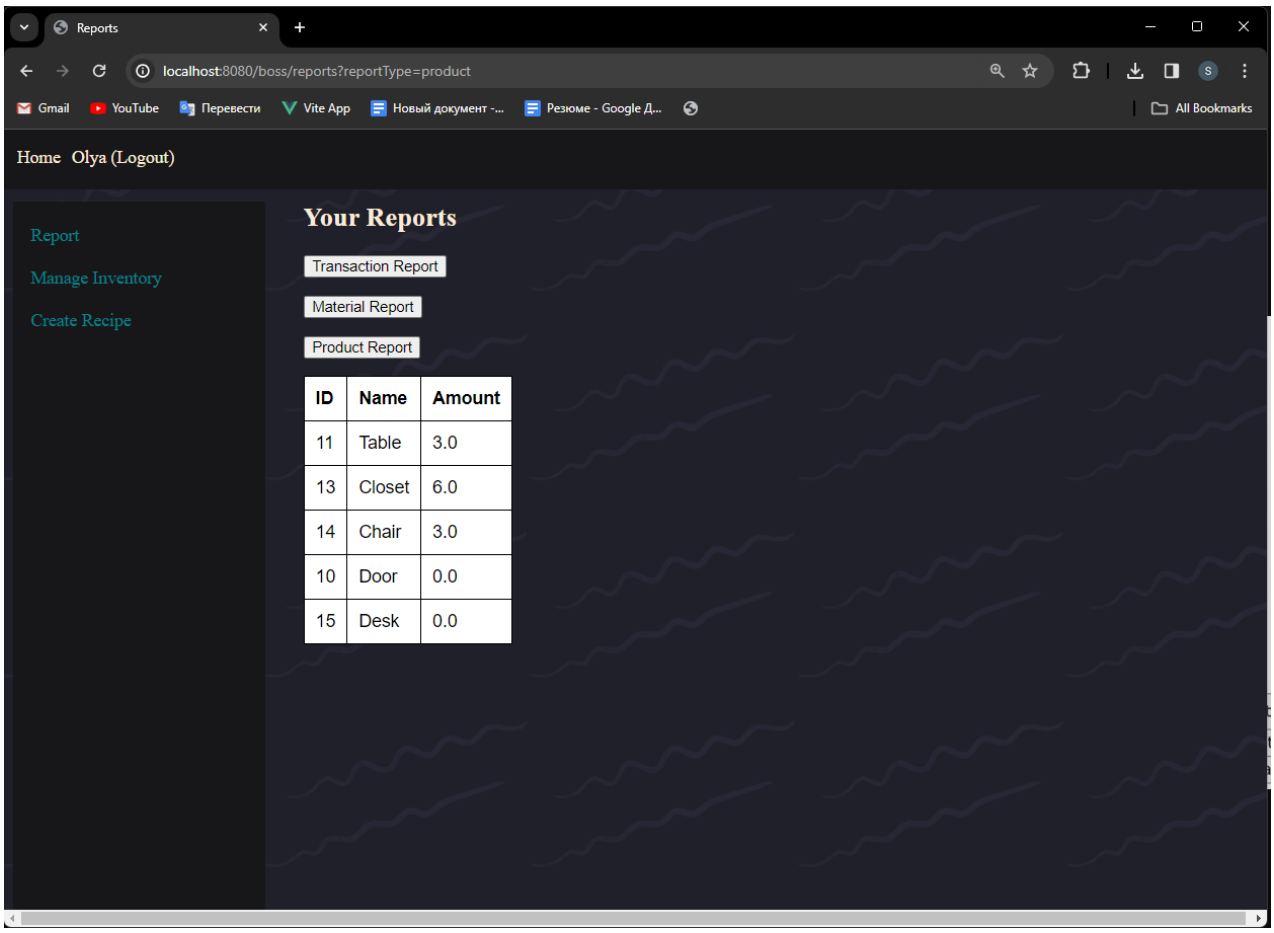
Приклад використання в контролері:

```java
@Autowired
private MaterialReportStrategy materialReportStrategy;
@Autowired
private ProductReportStrategy productReportStrategy;
@Autowired
private TransactionReportStrategy transactionReportStrategy;

 serrb *
@GetMapping("/boss/reports")
private String viewReports(Model model,
                           @RequestParam(value = "reportType", required = false) String reportType) {
    if (reportType == null) {
        reportType = "transaction";
    }
    ReportStrategy reportStrategy;
    switch (reportType) {
        case "material":
            reportStrategy = materialReportStrategy;
            break;
        case "product":
            reportStrategy = productReportStrategy;
            break;
        case "transaction":
        default:
            reportStrategy = transactionReportStrategy;
            break;
    }

    model.addAttribute( attributeName: "report", reportStrategy.generateReport());
    model.addAttribute( attributeName: "reportType", reportType); // Додано
    return "/boss/reports";
}
```

Приклад на сторінці Користувача (Начальник складу):







Висновок: на цій лабораторній роботі я познайомився з такими паттернами, як «singleton», «iterator», «proxy», «state», «strategy», розібрався у принципі їх роботи та сфері використання. На практиці спроектував та реалізував паттерн «Strategy» у своєму проекті.