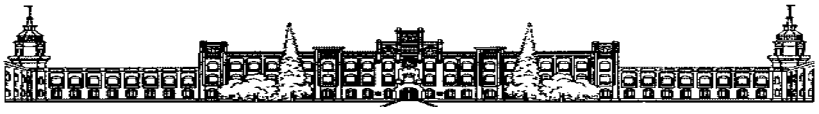

Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №5

Технології розроблення програмного забезпечення

ШАБЛОНИ «ADAPTER», «BUILDER», «COMMAND», «CHAIN OF RESPONSIBILITY», «PROTOTYPE»

Особиста бугалтерія

Виконав
студент групи ІА – 21:
Ліщинський Б. В.

Перевірив:
Амонс О. А.

Київ 2024

Тема: шаблони «ADAPTER», «BUILDER», «COMMAND», «CHAIN OF RESPONSIBILITY», «PROTOTYPE»

Мета: Реалізувати частину функціоналу робочої програми та застосувати один з розглянутих шаблонів при реалізації програми.

Хід роботи

Згідно з варіантом завдання та діаграмою класів, побудованою у ході виконання лабораторної роботи №2(рис. 1), при реалізації рівня сервісів програми було використано шаблон проектування “Prototype”.

Патерн Prototype — це структурний патерн проектування, який дозволяє створювати нові об'єкти шляхом копіювання вже існуючих. Замість того, щоб використовувати конструктори для створення нових екземплярів класу, ви можете викликати метод `clone()` у існуючого об'єкта, що спрощує процес і зменшує витрати ресурсів.

Цей патерн корисний, коли об'єкти мають спільні характеристики, або коли створення нового об'єкта є затратним за часом і ресурсами. Наприклад, уявіть клас автомобіля, який має багато параметрів. Замість повторного налаштування кожного разу, ви можете клонувати базовий автомобіль і змінювати лише певні властивості.

Основними перевагами патерну Prototype є зменшення витрат на створення об'єктів і гнучкість у проектуванні. Однак реалізація може бути складною, особливо при глибокому клонуванні, що може призвести до неявних зв'язків між об'єктами.

У ході виконання лабораторної роботи реалізував мікросервіси та gateway для доступу до цих мікросервісів

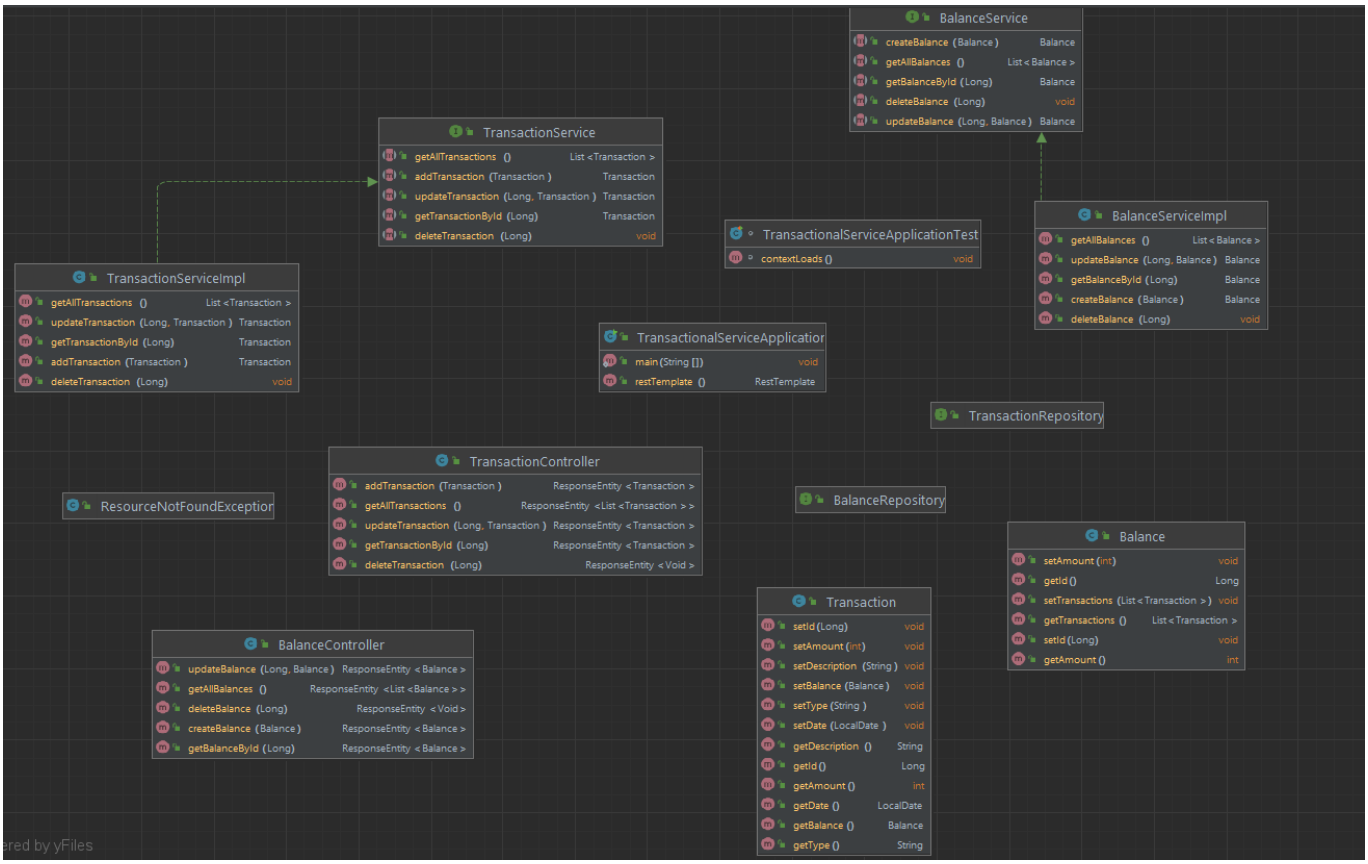

Рисунок 2 – Дерево програмного коду проекту транзакцій

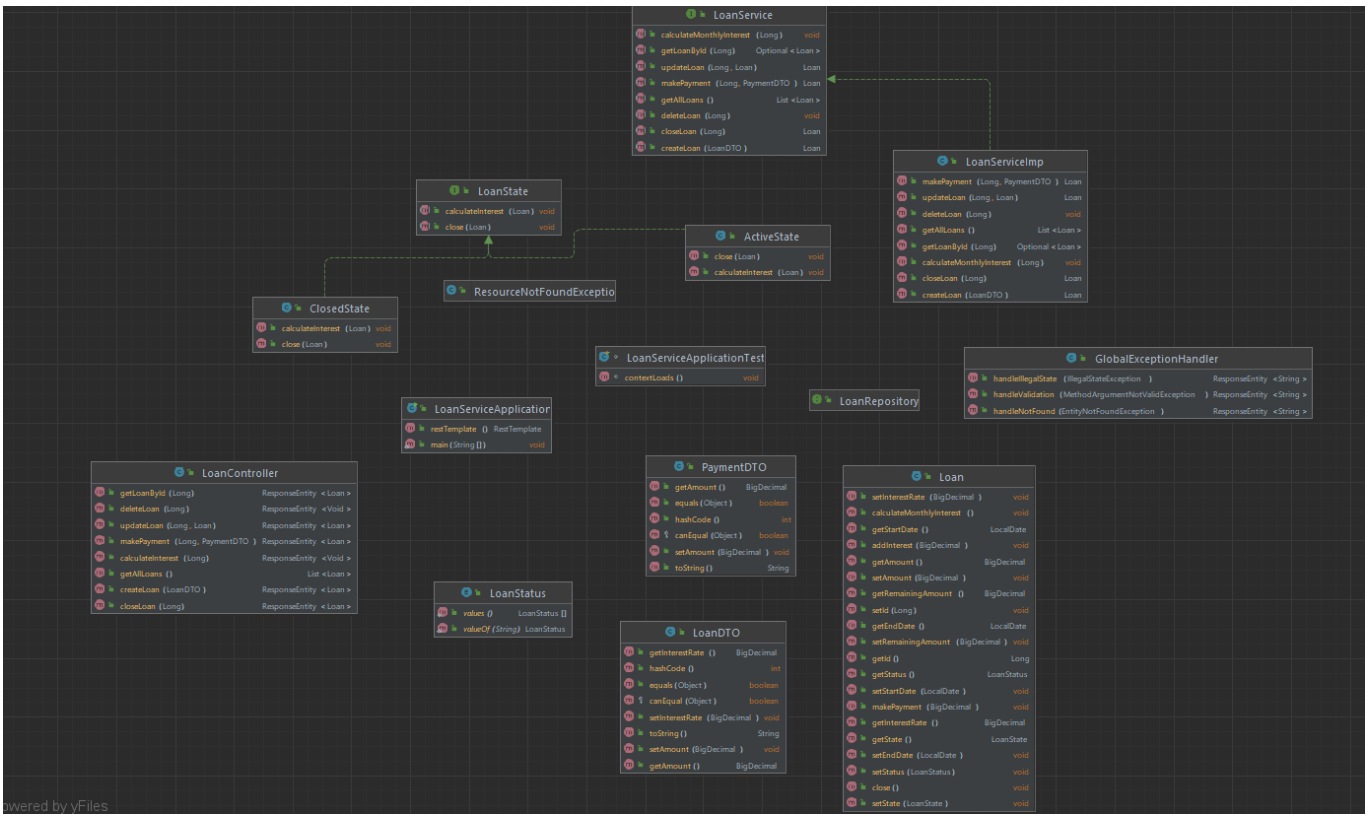

Рисунок 3 – Дерево програмного коду проекту кредитів

```yaml
server:
  port: 8080

spring:
  application:
    name: API-GATEWAY
  cloud:
    gateway:
      routes:
        - id: TRANSACTIONAL-SERVICE
          uri: lb://TRANSACTIONAL-SERVICE
          predicates:
            - Path=/processing/**
        - id: LOAN-SERVICE
          uri: lb://LOAN-SERVICE
          predicates:
            - Path=/loans/**

eureka:
  client:
    register-with-eureka: true
    fetch-registry: true
    service-url:
      defaultZone: http://localhost:8761/eureka/
  instance:
    hostname: localhost
```

Рисунок 4 – Точка доступу до сервісів (Налаштування gateway)

Знаходиться на порту 8080 та отримує доступ до мікросервісів.

Реалізація патерну State

```java
public interface LoanState {
    1 usage   2 implementations
    void close(Loan loan);

    1 usage   2 implementations
    void makePayment(BigDecimal payment, Loan loan);

    1 usage   2 implementations
    void calculateInterest(Loan loan);

}
```

```java
3 usages
public class ActiveState implements LoanState {
    1 usage
    @Override
    public void close(Loan loan) {
        if (loan.getRemainingAmount().compareTo(BigDecimal.ZERO) <= 0) {
            loan.setState(new ClosedState());
            loan.setStatus(LoanStatus.CLOSED);
        } else {
            throw new IllegalStateException("Cannot close loan with remaining amount: "+ loan.getAmount());
        }
    }
    1 usage
    @Override
    public void makePayment(BigDecimal payment, Loan loan) {
        loan.setRemainingAmount(loan.getRemainingAmount().subtract(payment));
    }

    1 usage
    @Override
    public void calculateInterest(Loan loan) {
        double interest = loan.getRemainingAmount().doubleValue() *
                loan.getInterestRate().doubleValue() / 12;
        double roundedInterest = Math.round(interest * 100.0) / 100.0;
        loan.addInterest(BigDecimal.valueOf(roundedInterest));
    }
}
```

```java
1 usage
public class ClosedState implements LoanState {
    1 usage
    @Override
    public void close(Loan loan) {
        throw new IllegalStateException("Loan is already closed");
    }

    1 usage
    @Override
    public void makePayment(BigDecimal payment, Loan loan) {
        throw new IllegalStateException("Cannot make payment on closed loan");
    }

    1 usage
    @Override
    public void calculateInterest(Loan loan) {
        throw new IllegalStateException("Cannot calculate interest for closed loan");
    }
}
```

Рисунок 5,6,7 реалізація патерну State

Loan використовує екземпляр LoanState, як статус для крелиту, відносто його типу виконуються різні дії при однакових запитах. До прикладу, ми не зможемо закрити кредит, якщо він ще не погашений, навіть якщо попросимо сервер зробити це.

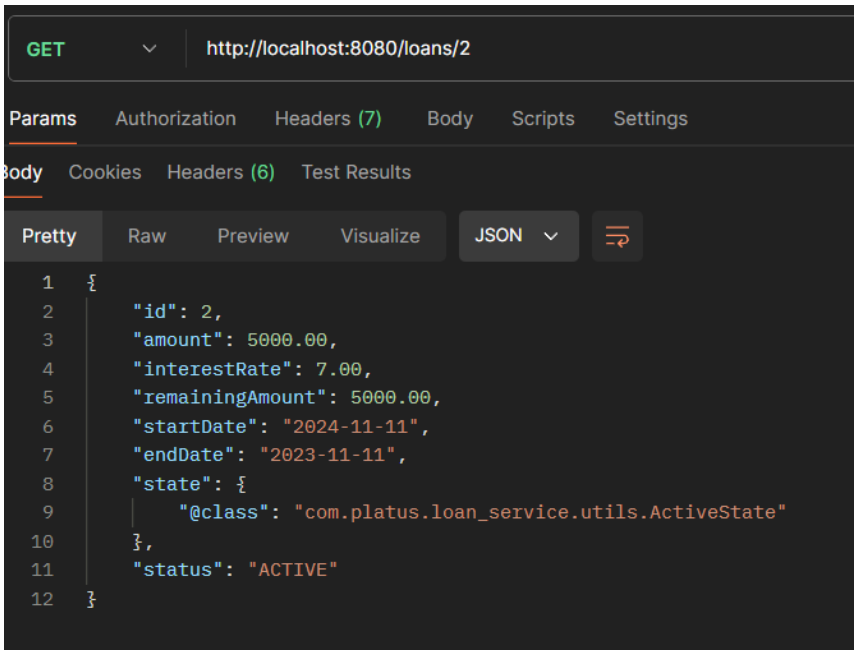

Рисунок 8 – створення кредиту

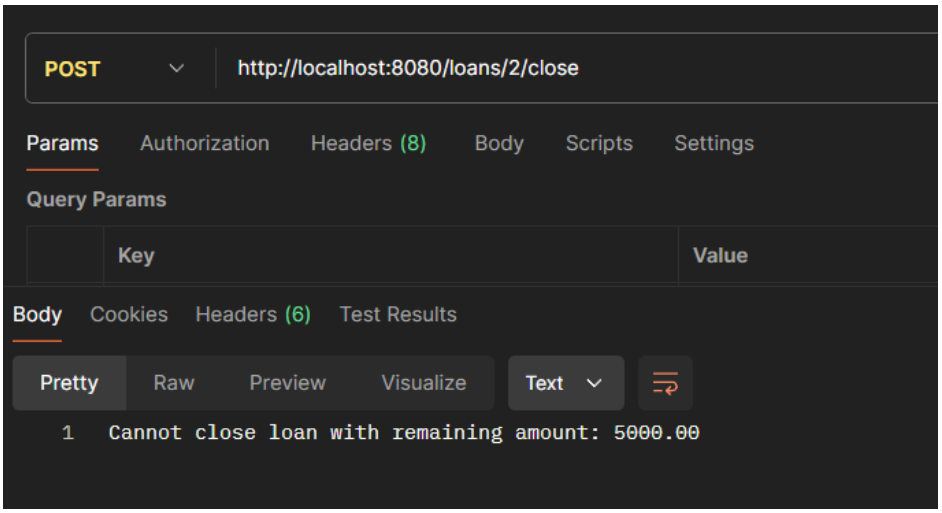

Рисунок 9 – спроба закрити кредит якщо він непогашений

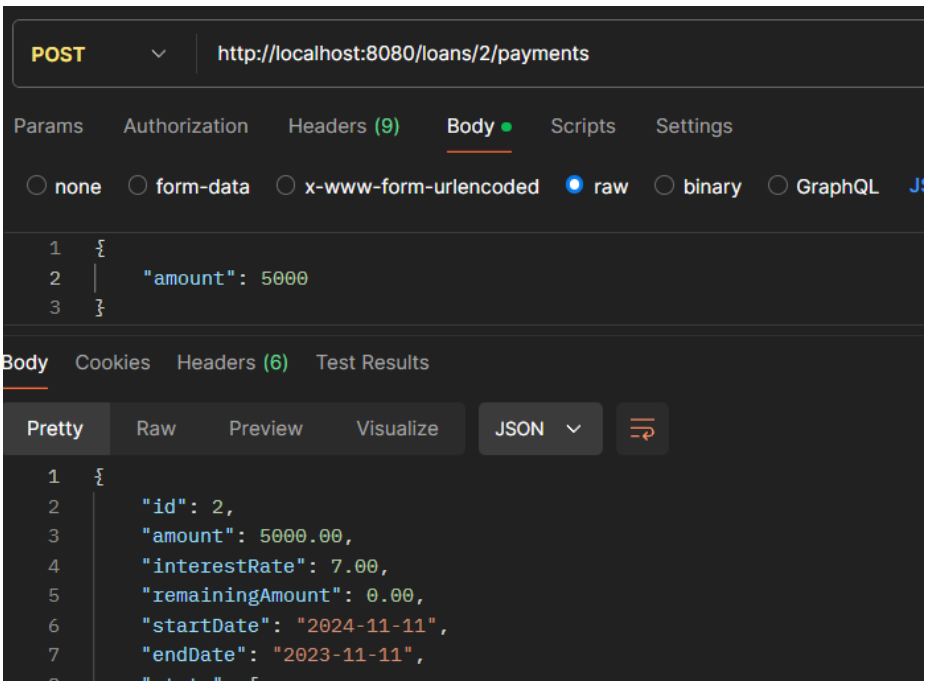

Рисунок 10 – Оплата кредиту

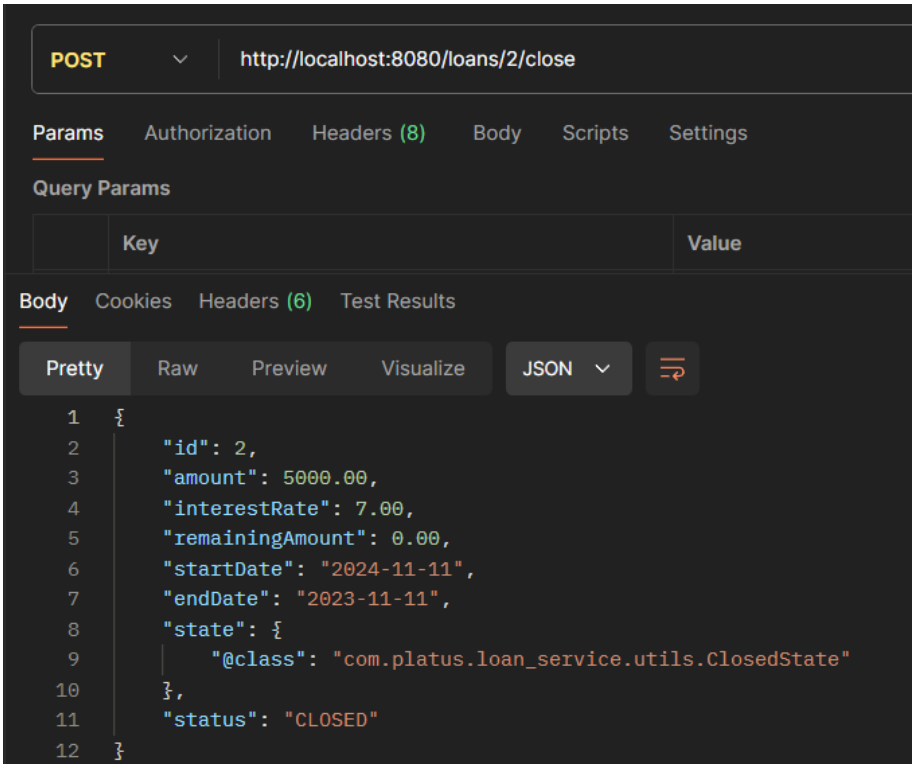

Рисунок 11 – Успішне закриття кредиту

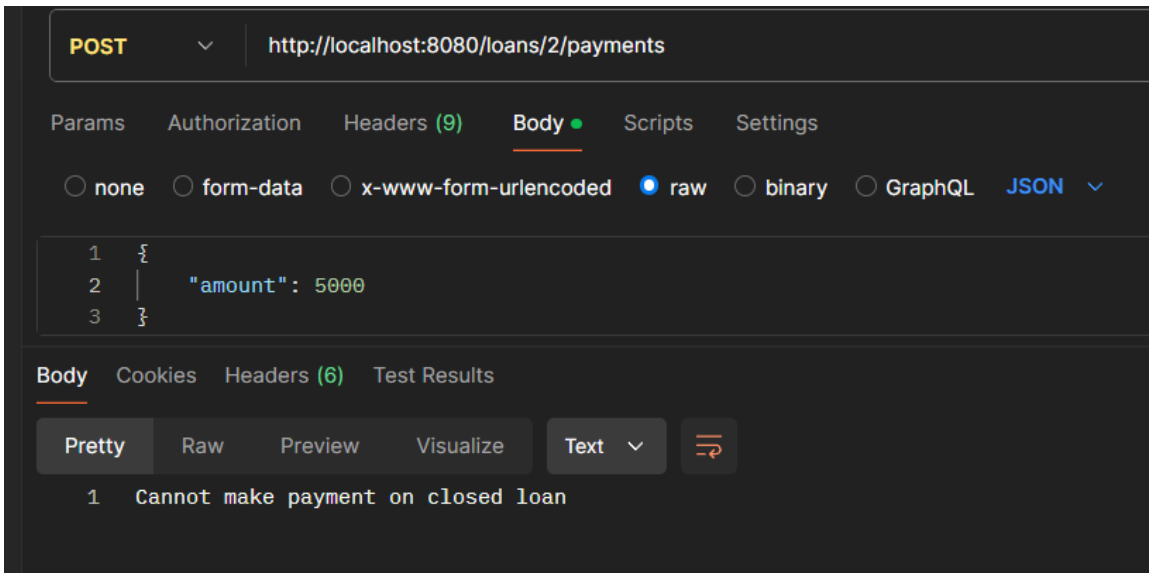

Рисунок 12 – Спроба зробити платіж на закритий кредит

Висновки: я реалізував частину функціональності системи, застосувавши шаблон проектування.