УДК 004.6

Поплавський А. В., ст. магістратури факультету кібернетики;
науковий керівник – д.ф.-м.н. професор Джунь Й. В. (Міжнародний економіко-гуманітарний університет імені академіка Степана Дем'янчука, м. Рівне)

ШАБЛОН ПРОЕКТУВАННЯ «REPOSITORY», ЯК ОДИН З НАЙБІЛЬШ УНІВЕРСАЛЬНИХ ІНСТРУМЕНТІВ ПРИ ПРОЕКТУВАННІ СКЛАДНИХ WEB СЕРВІСІВ

Анотація. *У статті досліджено основні переваги та практичні аспекти використання шаблону проектування «Репозиторій» при створенні середніх та великих за обсягом програмних продуктів. Обґрунтовано доцільність його вивчення та практичного застосування розробниками програмного забезпечення всіх кваліфікаційних рівнів. Розглянуто приклад застосування цього шаблону при створенні функціоналу оплати через сайт з використанням декількох платіжних систем.*
Ключові слова: *Шаблони проектування, ООП, Паттерн «Репозиторій», програмування, SOLID.*

Abstract. *The article explores the main advantages and practical aspects of using the «Repository» design template when creating medium and large-scale software products. The expediency of its study and practical application by software developers of all qualification levels is substantiated. An example of the application of this template in the creation of payment functionality through the site using several payment systems is given.*
Keywords: *Design patterns, OOP, Repository Pattern, programming, SOLID.*

Шаблон проектування «Репозиторій» – це дуже корисний інструмент при створенні складних програмних продуктів з можливістю широкого застосування в різних мовах програмування. Це один з найпопулярніших шаблонів проектування в наш час. Він дотримується твердих принципів SOLID і простий у використанні.

Актуальність обраної теми дослідження полягає в тому, що Інтернет-сайти вже давно перестали бути простими сторінками де використовується в класичному вигляді html+css, js і php+MySql. Нині це складні інфраструктурні сервіси, які поєднують в собі багато технологій, фреймворків, можуть мати декілька баз даних, в тому числі на AWS, взаємодіють з API інших веб-сервісів, та самі, як правило, складаються з декількох структурних елементів. Розробник програмного забезпечення приходячи на нове місце роботи часто не розуміє, що робиться в програмному коді, якщо той напи-

саний без використання будь-яких шаблонів проектування, і структуру цієї системи може зрозуміти тільки сам автор. Написання тестів для такого продукту надзвичайно ускладнюється, або взагалі стає неможливим, а розширення функціоналу породжує багато нових багів, які важко виявити.

Використання «Репозиторію» як шаблону проектування є нині новою темою, яка лише набирає популярність, тому її дослідження науковцями можна зустріти лише в роботах Метта Зандстри [1], Мартіна Фаулера та Кайла Галбрайтха. Перше згадування про нього було в книзі Еріка Еванса «Domain-Driven Design: Tackling Complexity in the Heart of Software» [2].

Метою нашої роботи є дослідження можливих сценаріїв застосування «Репозиторію», як шаблону проектування.

Шаблон проектування Repository є посередником між шаром області визначення і шаром розподілу даних, працюючи, як звичайна колекція об'єктів області визначення. Об'єкти-клієнти створюють опис запиту декларативно і направляють їх до об'єкта-сховища (Repository) для обробки. Об'єкти можуть бути додані або видалені з сховища, як ніби вони формують просту колекцію об'єктів. А код розподілу даних, прихований в об'єкті Repository, подбає про виконання відповідних операторів в непомітно для розробника. Шаблон проектування Repository інкапсулює об'єкти, які знаходяться в сховищах даних і операції, виконувані над ними, надаючи більш об'єктно-орієнтоване уявлення реальних даних. Repository також має на меті досягнення повного поділу і односторонньої залежності між рівнями області визначення і розподілу даних. Основна перевага такого підходу – абстракція, яку Repository забезпечує. Це означає, що він додає ще один шар між логікою програми та даними, таким чином абстрагується від конкретного постачальника даних будь то ORM, API стороннього продукту, додатковий сервіс свого продукту, чи фейкові дані під час виконання тестів [3].

Основна ідея цього шаблону – створити загальний абстрактний спосіб роботи програми з рівнем даних, не турбуючись, якщо реалізація буде спрямована на локальну базу даних, БД на віддаленому сервері, або на API стороннього ресурсу [2]. Repository інкапсулює набір об'єктів, що зберігаються в сховищі даних, і операції, що виконуються над цими обєктами, забезпечуючи більш об'єктно-орієнтоване уявлення реальних даних. Repository також має на меті досягнення повного поділу і односторонньої залежності між рівнями області визначення і розподілу даних.

Інколи Repository помилково називають реалізацію іншого патерну – DAO (Data Access Object). Або обидва цих паттерна реалізуються одним і тим же класом, але відмінність в тому, що DAO – об'єкт який надає абстрактний інтерфейс до деяких видів баз даних чи механізмів персистентності реалізуючи певні операції без розкриття деталей бази даних. Він надає відображення від програмних викликів до рівня персистентності. В той же

час Repository являє собою колекції об’єктів. Вони не описують зберігання в базах даних або кешування або рішення будь-якої іншої технічної проблеми. Репозиторії представляють колекції. Як зберігати ці колекції – це просто деталь реалізації. Основна перевага Repository – це абстрактний механізм зберігання для колекцій сутностей. Надавши інтерфейс «Репозиторію», тим самим розв’язуються руки розробнику, який сам вирішує як і де зберігати дані [4].

В різних статтях зустрічаються дискусії з приводу сервісного прошарку, на якому знаходиться Application Layer чи Domain Layer. Для початку необхідно визначити, що Application Service Layer – це багаторівнева архітектура, яка відповідає за специфічні деталі реалізації програми, такі як цілісність бази даних, і різні реалізації роботи з інтернет-протоколами (відправка електронної пошти, API) і ін. А Domain Layer це прошарок багаторівневої архітектури, яка відповідає за бізнес-правила і бізнес-логіку сервісу. Це означає, що інтерфейс знаходиться на межі шарів. і насправді може містити доменно-специфічні концепти, але сама реалізація не повинна цього робити.

Інтерфейси «Репозиторіїв» належать до шару домену. Їх реалізація відносяться до Domain Layer. Це означає, що розробники вільні при побудові архітектури на рівні Domain Layer без необхідності залежати від Application Service Layer. Шаблон проектування Repository став популярним завдяки DDD (Domain Driven Design). На противагу до Database Driven Design в DDD його розробка починається з проектування бізнес логіки, беручи до уваги тільки особливості предметної області та ігноруючи все, що пов’язано з особливостями бази даних або інших способів зберігання даних. Спосіб зберігання бізнес об'єктів реалізується в другу чергу.

Для кращого розуміння, можна розглянути принцип роботи цієї схеми на прикладі SaaS сервісу, або інтернет магазину, який працює з різними платіжними системами (рис. 1).

При такому способі реалізації можна отримати однакові дані незалежно від того, в який спосіб користувач здійснив платіж. При цьому, кожен із API клієнтів платіжних систем зобов’язаний реалізувати методи описані в PaymentInterface, і, якщо виникне необхідність, додати ще одну або декілька платіжних систем. Це без проблем можна буде зробити, при тому бути впевненим, що жоден інший функціонал не зламається. Також, в цьому випадку, зручно писати Feature тести для PaymentController, оскільки можна реалізувати імітацію API клієнта платіжної системи, який буде генерувати і повертати фейкові дані про платежі TestPaymentSystem (рис. 1), під час тестування його необхідно лише передати в конструктор PaymentRepository.

Практичну реалізацію схеми на рис. 1 можна представити в наступному PHP коді:

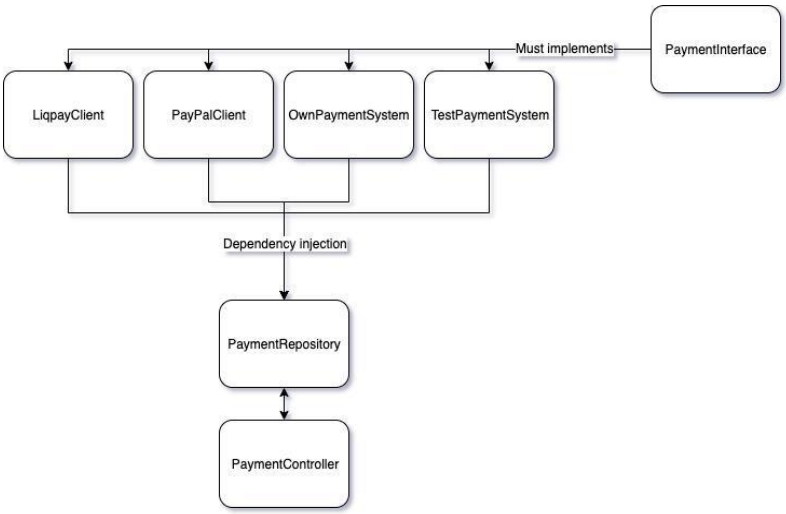

Рис. 1. Схема використання шаблону проектування «Repository»

```
interface PaymentInterface
{public function pay(Card $card): string;
public function payStatus(string $transactionId): bool;
public function findById(string $transactionId): array;}
```

Цей інтерфейс зобов'язує реалізувати ці 3 методи в кожен клас який його імплементує. Метод pay (Card $card) для здійснення оплати, який приймає дані платіжної картки і акумулює дані типу string, наприклад ID транзакції. Метод payStatus(string $transactionId) для перевірки статусу виконання платежу, який зобов'язаний повернути true/false. Метод findById (string $transactionId), який повертає дані про транзакцію в вигляді масиву:

```
class PayPalClient implements PaymentInterface
{public function pay(Card $card): string
{$transactionId = // go to PayPall and try to pay
return $transactionId;}
public function payStatus(string $transactionId): bool
{$status = // go to PayPall and check status by
transactionId return $status;}
public function findById(string $transactionId): array
{$data = // go to PayPall and get data by transactionId
return $data;}
```

Клас клієнт платіжної системи виконує функцію взаємодії лише з однією платіжною системою через API, де кожен метод класу може виконувати лише одну функцію, і повинен повертати значення, яке строго відповідає типу описаному в інтерфейсі який цей клас реалізує.

```
class PaymentRepository {private $paymentSystem;
public function __construct(PaymentInterface
$paymentSystem)
{$this->paymentSystem = paymentSystem;}
public function makePayment(Card $card): array
{$transactionId = $this->paymentSystem->pay($card);
if(!$this->paymentSystem->payStatus($transactionId))
{throw new Exception();}
return $this->paymentSystem->findById($transactionId);}
```

Клас PaymentRepository може прийняти в конструкторі лише клас, який реалізує інтерфейс PaymentInterface. При виклику класу PaymentRepository впровадити залежність PaymentInterface можна двома способами:

1. Оголосити новий екземпляр класу PaymentRepository, передавши в конструктор необхідний клас реалізуючий PaymentInterface;

2. Зареєструвати його за допомогою IoC контейнера в необхідному сервіс провайдері, в тому випадку якщо передбачено використання сервіс провайдера.

Аналогічний підхід також зручно використовувати при авторизації користувача, якщо передбачається вхід на сайт за допомогою інших сайтів, використовуючи спосіб авторизації OAuth, наприклад: google, facebook, linkedin, github, або ввести логін/email, пароль в відповідні поля форми, таким чином увійти на сайт.

Основними перевагами використання Repository можна назвати такі:

1. Виділення загальної логіки (DRY): перевірки, значення за замовчуванням, логування, і т.п.;

2. Незалежність бізнес-логіки від способу зберігання. Використовуючи Repository, можна зустрітися з колекціями бізнес об'єктів (POCO), але не з database related об'єктами, ні з Data Access Objects. Можливість використовувати різні способи зберігання: ORM, rdbms, cloud storage, file system etc, взаємозамінювати і комбінувати їх;

3. Працюючи через інтерфейси можна створити декілька реалізацій сховища. Це допомагає при тестуванні (можна передавати заглушку сховища при тестуванні бізнес-логіки) і при зміні способу зберігання даних. Конкретна реалізація сховища може реєструватися в IoC контейнері, і таким чином жоден модуль програми за винятком точки входу, не буде пов'язаний з мо-

дулем реалізації сховища. Додаток буде працювати з інтерфейсом сховища та об'єктами бізнес-логіки.

За результатами проведеного дослідження можна зробити висновок, що шаблон проектування Repository, є універсальним інструментом, при розробці великих і середніх за розміром веб-сервісів, а також зрозумілим за своєю логікою роботи, що суттєво допомагає розробникам програмного забезпечення при необхідності масштабування програмного продукту.

1. Мэтт Зандстра PHP: объекты, шаблоны и методики программирования, 5-е изд. 2019. 736 с. **2.** Repository Design Pattern. URL:https://medium.com/@pererikbergman/repository-design-pattern-e28c0f3e4a30 (дата звернення: 11.10.2019) **3.** Справочник «Паттерны проектирования». URL: http://design-pattern.ru/patterns/repository.html (дата звернення: 11.10.2019) **4.** EntityFramework: (анти)паттерн Repository. URL: https://habr.com/ru/post/335856/ (дата звернення: 11.10.2019).