№ 1(54) 2025

МЕТОДИ ТА ПРИЛАДИ КОНТРОЛЮ ЯКОСТІ

ISSN 1993-9981 print
ISSN 2415-3575 online

https://mpky.nung.edu.ua

Комп'ютерні технології та системи

Прийнято 09.06.2025. Прорецензовано 20.06.2025. Опубліковано 27.06.2025.

УДК 004.054 DOI: 10.31471/1993-9981-2025-1(54)-156-163

ВИКОРИСТАННЯ ПАТЕРНІВ ПРОГРАМУВАННЯ В АВТОМАТИЗОВАНОМУ ТЕСТУВАННІ

Гарасимів В. М.*
Кандидат технічних наук, доцент
Івано-Франківський національний технічний університет нафти і газу
76019, вул. Карпатська, 15, м. Івано-Франківськ, Україна
https://orcid.org/0000-0002-6613-3549
e-mail: vira.harasymiv@nung.edu.ua

Анотація. У статті розглядаються особливості побудови проєктів, що реалізують шаблони проєктування Page Object та Page Factory для створення автоматизованих тестів. Основною мовою програмування обрано Java, яка належить до об'єктно-орієнтованої парадигми, а розробка проєктів здійснювалася в середовищі IntelliJ IDEA. У фокусі – тестування процесу реєстрації нового користувача на веб-ресурсі. Було реалізовано відповідні автотести з використанням зазначених шаблонів проєктування разом із бібліотеками Selenium-java, WebDriverManager і тестовим фреймворком TestNG. Дані, необхідні для успішної реєстрації, зчитуються з конфігураційного файлу у форматі .properties, який зберігається у директорії resources. Результатом тестування є згенерований звіт, що демонструє успішність проходження автотесту. Кожна веб-сторінка представлена окремим класом, у якому реалізовано методи для взаємодії з нею. У випадку використання лише Page Object, елементи інтерфейсу приховані (мають модифікатор private) та винесені в окремий модуль. Методи сторінок повертають нові екземпляри класів, що відповідають подальшим етапам сценарію тестування. Код автотестів є логічно відокремленим від елементів інтерфейсу та методів, які моделюють дії користувача. Також, завдяки використанню анотацій @BeforeSuite, @AfterSuite та @BeforeClass, створено методи, що відповідають за підготовку та завершення виконання тестових наборів. Подано порівняння структур проєктів, побудованих за шаблонами Page Object та Page Factory. Оскільки Page Factory є більш оптимізованим підхід для ініціалізації елементів сторінки або об'єкта сторінки, це дозволяє зробити структуру проєкту простішою. У цьому випадку, елементи інтерфейсу розміщуються безпосередньо у відповідних класах сторінок.

Ключові слова: проєкт, шаблон проєктування, Page Object, Page Factory, автотест, елементи інтерфейсу, веб-застосунок, тестовий фреймворк.

Вступ

Створення автоматизованих тестів є ключовим засобом виявлення помилок на початкових етапах життєвого циклу програмного продукту. Завдяки впровадженню автоматизованого тестування з'являється можливість знизити витрати на усунення помилок і підвищити ефективність процесу контролю якості. Незважаючи на те, що розробка автотестів

Запропоноване посилання: Гарасимів, В. М. (2025). Використання патернів програмування в автоматизованому тестуванні. Методи та прилади контролю якості, 1(54), 156-163. doi: 10.31471/1993-9981-2025-1(54)-156-163

* Відповідальний автор

може видаватися простою справою для програмістів або тестувальників, існує висока ймовірність того, що тестові сценарії будуть реалізовані нераціонально, що ускладнює їх обслуговування. Часто буває, що зміна одного елементу на веб-сторінці, який використовується у багатьох тестах, змушує переглянути та модифікувати увесь набір тестових скриптів. Це призводить до значних витрат часу і знижує доцільність використання автотестів на ранніх фазах розробки програмного забезпечення [1]. Саме тому з'явилися різні шаблони проєктування, які сприяють створенню легко підтримуваних та повторно використовуваних автоматизованих тестів.

Мета роботи

Page Object є одним із найрозповсюдженіших архітектурних підходів у сфері автоматизованого тестування. Цей шаблон проєктування дозволяє ізолювати взаємодію з елементами веб-сторінки, що сприяє зменшенню обсягу коду та спрощенню його супроводу. Під час реалізації проєкту за допомогою патерна Page Object важливо відокремлювати автотестові сценарії від класів сторінок, які містять методи взаємодії з відповідними елементами. Самі веб-елементи слід зберігати окремо. З упровадженням патерна Page Factory процес створення автотестів став зручнішим, оскільки він дозволяє зберігати веб-елементи безпосередньо в класі сторінки, що змінює загальну структуру проєкту. Таким чином, мета цієї роботи – проаналізувати специфіку структурування проєкту з урахуванням особливостей застосування патернів Page Object і Page Factory.

Теоретичне обґрунтування

На сучасному етапі більшість програмного забезпечення створюється із застосуванням гнучких методологій та інкрементально-ітеративних моделей розробки, що ґрунтуються на постійному внесенні змін і вдосконалень [2]. Такий підхід зумовив зростання обсягу регресійного тестування, що у свою чергу, зробило ручну перевірку малоефективною та трудомісткою. У зв'язку з цим автоматизація тестування програмних продуктів набула широкого розповсюдження та стала важливою складовою забезпечення якості.

Для покращення процесу розробки та обслуговування автоматизованих тестів використовуються різноманітні шаблони проєктування. Найбільш уживаними серед них вважаються Page Object, Page Factory і Page Element [3]. Основна складність при роботі з цими шаблонами полягає у правильному їх впровадженні, а також у чіткому усвідомленні того, що вибір конкретного патерну повинен відповідати вимогам конкретного завдання та сприяти його ефективному вирішенню [4].

Одним із найбільш поширених архітектурних підходів у сфері автоматизованого тестування є шаблон проєктування Page Object. Застосування цього патерну дає змогу ізолювати логіку взаємодії з окремими елементами веб-сторінки, що дозволяє скоротити обсяг коду та спростити його супровід [5]. Під час реалізації проєктів з використанням цього підходу слід враховувати, що автотестові скрипти мають бути відокремленими від класів, які описують сторінки та містять відповідні функціональні методи. Елементи інтерфейсу (веб-елементи) зберігаються окремо від тестової логіки.

З появою шаблону Page Factory процес створення автотестів став більш зручним, оскільки цей підхід дозволяє розміщувати елементи сторінки безпосередньо в її відповідному класі. Такий підхід сприяє зміні архітектурної структури проєкту та спрощує його підтримку [6].

Таким чином, мета даного дослідження полягає у вивченні структурних особливостей побудови програмного проєкту з використанням зазначених шаблонів проєктування.

Методика експерименту

Припустимо, що об'єктом тестування виступає веб-ресурс за посиланням: http://practice.bpbonline.com/index.php. У

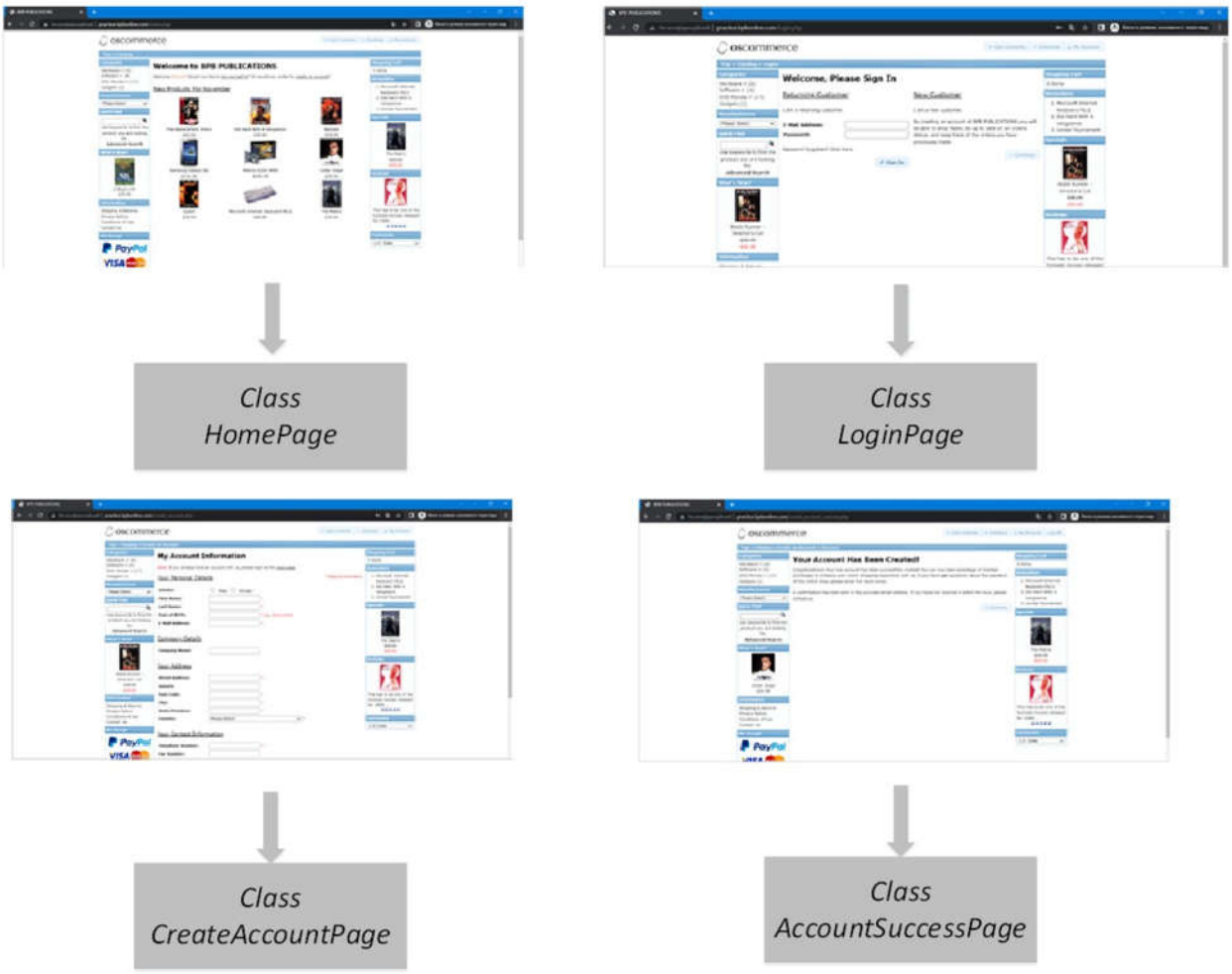

Рисунок 1 – Декомпозиція тестового випадку на класи програмування відповідно до патерну Page Object

якості прикладу розглядається тестовий сценарій, спрямований на перевірку успішного проходження процедури реєстрації користувача. Декомпозиція цього кейсу на відповідні класи (що представляють сторінки веб-застосунку) наведена на рисунку 1.

На першому етапі відбувається перехід на головну сторінку сайту, яка реалізована у вигляді класу HomePage. Далі, для ініціації реєстрації, обирається пункт меню «My Account», що перенаправляє користувача на сторінку авторизації — клас LoginPage. У секції «New Customer» здійснюється натискання кнопки «Continue», після чого відкривається сторінка для внесення персональних даних з метою створення нового облікового запису – AccountPage. Після заповнення необхідної інформації та її підтвердження, користувач автоматично перенаправляється на сторінку з повідомленням про успішне завершення реєстрації – AccountSuccessPage.

Для ініціалізації проєкту в середовищі IntelliJ IDEA до конфігураційного файлу POM були додані необхідні залежності, зокрема Selenium-java, TestNG та WebDriverManager. Організацію структури проєкту представлено на рисунку 2.

Базовий клас BasePage виступає спільним предком для всіх класів, що відповідають окремим сторінкам веб-застосунку. Його конструктор реалізує ініціалізацію екземплярів WebDriver та WebDriverWait [7].

Окрім того, до цього класу доцільно додавати методи, які реалізують очікування наявності певних елементів інтерфейсу:

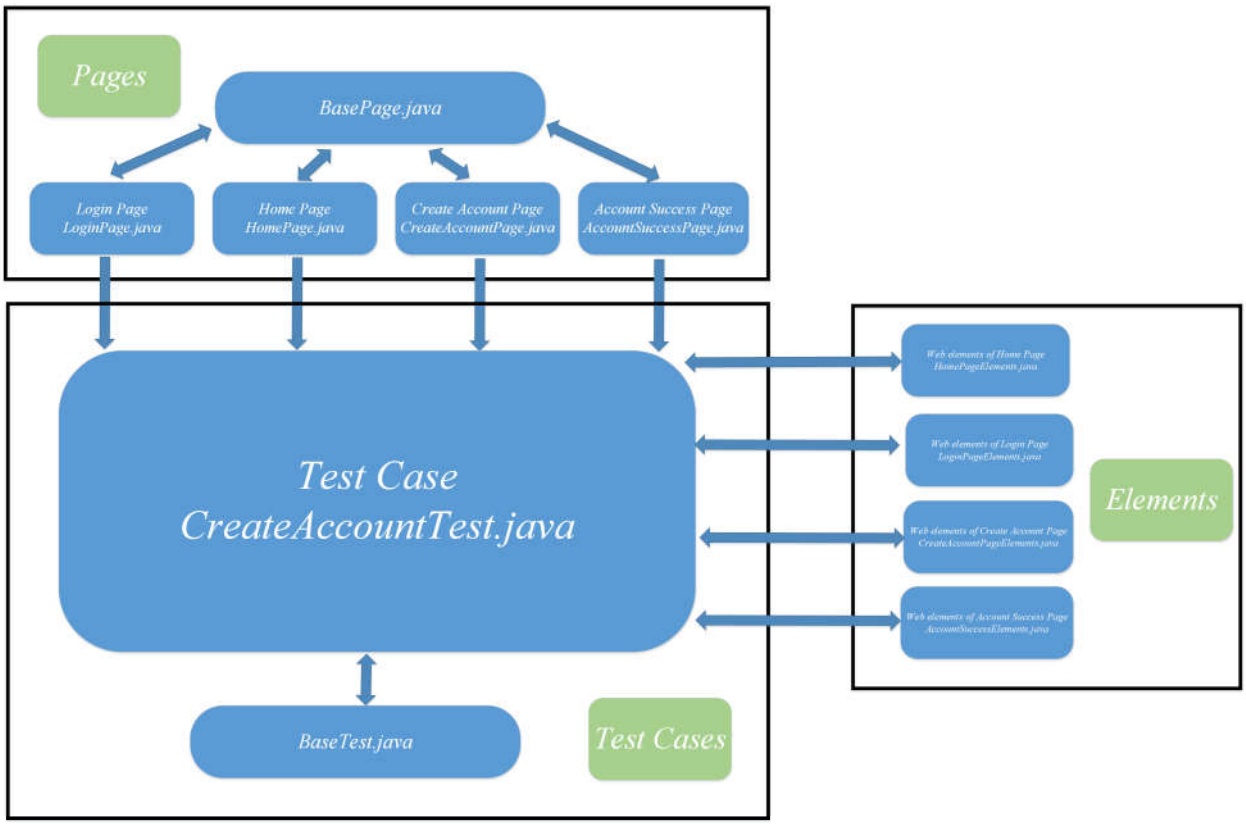

Рисунок 2 – Структурна схема проєкту із використанням патерну програмування Page Object

```
// BasePage.java
public abstract class AbstractPage {
    private static final int
DEFAULT_WAIT = 30;
    protected WebDriver browser;
    private WebDriverWait explicitWait;

    public AbstractPage(WebDriver
browser) {
        this.browser = browser;
        explicitWait = new
WebDriverWait(browser,
Duration.ofSeconds(DEFAULT_WAIT));
    }
    protected void waitUntilVisible(By
elementLocator) {
explicitWait.until(ExpectedConditions.
visibilityOfElementLocated(elementLocat
or));
    }
}
```

Клас HomePage включає метод clickOnMyAccountButton(), що імітує взаємодію користувача з елементом меню «My Account» та повертає об'єкт класу LoginPage, оскільки після цього відбувається перехід на відповідну сторінку:

```
// HomePage.java
public class StartPage extends
AbstractPage {
    public StartPage(WebDriver browser)
{
        super(browser);
    }
    public AuthPage navigateToLogin(By
accountMenuLocator) {

browser.findElement(accountMenuLocator)
.click();
        return new AuthPage(browser);
    }
}
```

Подібним чином, клас LoginPage містить метод clickOnContinueButton(), який відповідає за клік по кнопці «Continue». У класі CreateAccountPage реалізовано набір методів, які імітують послідовне заповнення полів форми реєстрації. Дані для заповнення зчитуються з конфігураційного файлу формату .properties, розміщеного у директорії resources.

Сторінка успішного завершення реєстрації представлена класом

Test	# Passed	# Skipped	# Retried	# Failed	Time (ms)	Included Groups	Excluded Groups
Smoke							
Creating new account test	1	0	0	0	8 162		

Class	Method	Start	Time (ms)
Smoke			
Creating new account test — passed			
com.IFNTUNG.edu.tests.CreateAccountTest	createAccountTest	1669586950220	2876

Creating new account test

com.IFNTUNG.edu.tests.CreateAccountTest#createAccountTest

Рисунок 3 – Звіт запуску автоматизованого автотесту для перевірки успішності створення нового користувача на сайті

AccountSuccessPage, що містить метод getActualMessage(), який зчитує текстове повідомлення після підтвердження реєстраційних даних

Сторінка успішного завершення реєстрації представлена класом AccountSuccessPage, що містить метод getActualMessage(), який зчитує текстове повідомлення після підтвердження реєстраційних даних.

```
// AccountSuccessPage.java
public class RegistrationCompletePage
extends AbstractPage {

    public
RegistrationCompletePage(WebDriver
browser) {
        super(browser);
    }

    public                          String
retrieveConfirmationMessage(By
messageLocator) {

waitUntilVisible(messageLocator);
        return
browser.findElement(messageLocator).get
Text();
    }
}
```

Базовий клас BaseTest містить усі спільні налаштування та ресурси, що використовуються у тестових класах-нащадках [8].

З метою уникнення помилок при ініціалізації елементів інтерфейсу, всі веб-елементи розміщуються окремо від класів сторінок, оголошуються приватними та використовуються через спеціальні методи доступу.

```
// BaseTest.java
public class CommonTestSetup {
    protected                  WebDriver
browserInstance;
    private    static    final    String
TARGET_URL                          =
"http://practice.bpbonline.com/index.ph
p";
    @BeforeSuite
    public void prepareEnvironment() {

WebDriverManager.chromedriver().setup();
    }
    @BeforeClass
    public void initializeDriver() {
        browserInstance      =      new
ChromeDriver();
        browserInstance.manage().
window().maximize();
       browserInstance.get(TARGET_URL);
    }
```

Клас CreateAccountTest реалізує тестовий сценарій, у якому перевіряється створення нового облікового запису. Тест вважається успішним у випадку, якщо після введення і підтвердження даних користувач отримає повідомлення про успішну реєстрацію. Звіт про результати виконання тесту, сформований фреймворком TestNG [9], представлено на рисунку 3.

У даному прикладі розглядається реалізація тестового сценарію з використанням шаблону проєктування Page Factory, який виступає розширенням шаблону Page Object. Цей підхід є більш оптимізованим та призначений для ініціалізації елементів веб-сторінки або створення екземпляру класу, що представляє відповідну сторінку. Основною відмінністю є те, що Page Factory використовує анотацію @FindBy для визначення веб-елементів, які

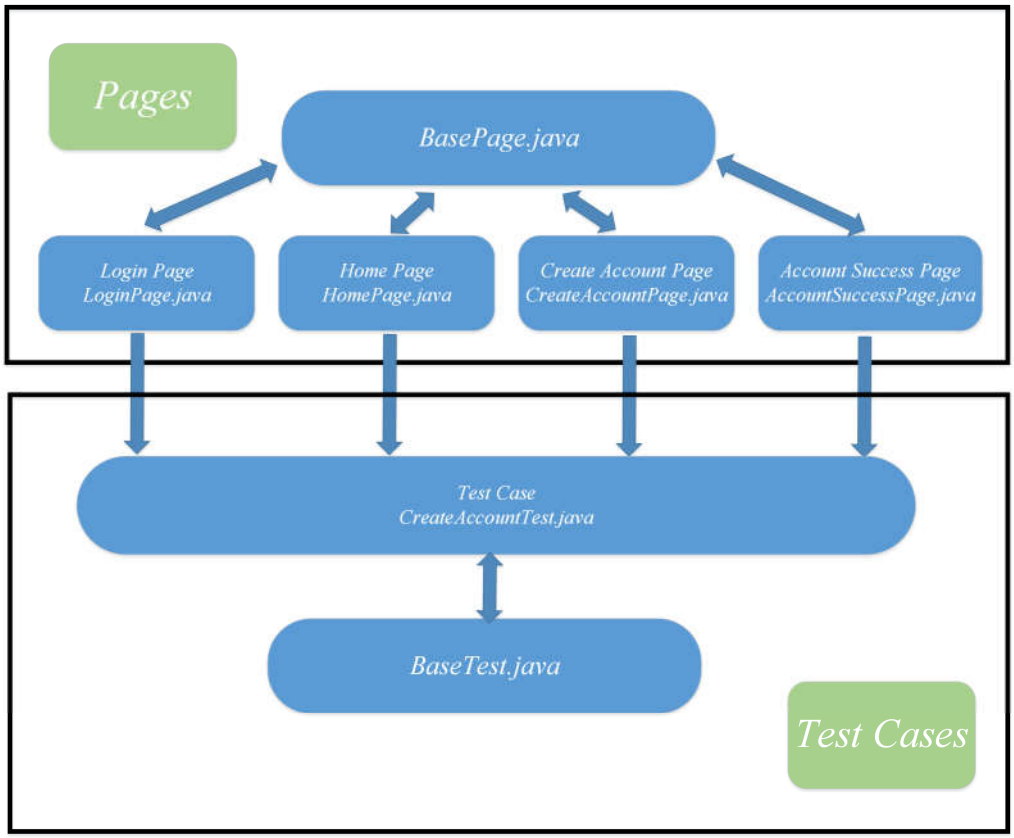

Рисунок 4 – Структурна схема проєкту з використанням патернів програмування Page Object та Page Factory

безпосередньо зберігаються в межах відповідного класу, а не винесені в окремі структури [10]. Такий підхід значною мірою спрощує розробку автотестів та позитивно впливає на організацію проєктної структури (рис. 4).

Для реалізації такого підходу необхідно модифікувати конструктор базового класу, забезпечивши ініціалізацію всіх елементів за допомогою PageFactory:

```
public abstract class AbstractPage {
    private static final int
WAIT_DURATION = 30;
    protected WebDriver browser;
    private WebDriverWait fluentWait;
    public AbstractPage(WebDriver
browser) {
        this.browser = browser;
        this.fluentWait = new
WebDriverWait(browser,
Duration.ofSeconds(WAIT_DURATION));

PageFactory.initElements(browser,
this); // Ініціалізація всіх елементів
за PageFactory
    }
```

```
    protected void waitUntil
ElementVisible(By elementLocator) {
        fluentWait.until(Expected
Conditions.visibilityOfElementLocated
(elementLocator));
    }
}
```

Висновки

У ході дослідження було розглянуто ключові підходи до структурування автоматизованих тестів із використанням патернів Page Object та Page Factory. Встановлено, що автоматизація тестування дозволяє своєчасно виявляти дефекти, знижуючи витрати на їх усунення на пізніших етапах життєвого циклу програмного забезпечення. Однак неструктуроване створення тестових сценаріїв може призвести до проблем у підтримці коду, особливо при змінах на стороні UI. Це підтверджує доцільність використання шаблонів проєктування, які забезпечують кращу модульність, ізольованість та повторне використання коду.

Патерн Page Object сприяє розділенню логіки тестування та опису веб-сторінок, що підвищує зручність розробки та обслуговування автотестів. Він дозволяє зберігати веб-елементи окремо від логіки взаємодії з ними, а також зменшує дублювання коду. Розширенням цього підходу є патерн Page Factory, який дозволяє інтегрувати веб-елементи безпосередньо в класи сторінок за допомогою анотацій @FindBy. Це значно спрощує структуру проєкту та покращує читабельність коду.

Під час експериментальної частини роботи було реалізовано тестовий сценарій реєстрації користувача на демонстраційному сайті, що дало змогу проілюструвати практичне застосування згаданих патернів. Було створено класи сторінок, які описують логіку взаємодії з елементами інтерфейсу, а також тести, які використовують ці класи. Архітектура проєкту була реалізована у середовищі IntelliJ IDEA з використанням бібліотек Selenium, TestNG та WebDriverManager.

Базовий клас BasePage забезпечив спільну функціональність для всіх сторінок, включаючи логіку очікування елементів. Класи окремих сторінок були реалізовані згідно з принципами патерну Page Object, а веб-елементи — проініціалізовані за допомогою анотацій Page Factory. Тестовий клас CreateAccountTest успішно перевірив сценарій реєстрації, підтверджуючи правильність реалізації.

Результати тестування були автоматично задокументовані за допомогою фреймворку TestNG. Отримані результати підтвердили, що застосування шаблонів Page Object та Page Factory підвищує надійність і масштабованість автотестів. Такий підхід забезпечує зручне розширення функціональності та зменшує витрати часу при оновленні тестів. З огляду на це, впровадження структурованих архітектур у процес автоматизації тестування є доцільним та ефективним рішенням.

Подяки
Відсутні.

Конфлікт інтересів
Відсутній.

Список використаних джерел / References

1. Selenium and TestNG. URL: https://testng.org/doc/selenium.html
2. Amir Ghahrai. Page Object Model Framework with Java and WebDriver. URL: https://devqa.io/page-object-framework-java-webdriver/
3. Selenium and TestNG. URL: https://testng.org/doc/selenium.html
4. Yoni Flenner. Page Object Model-Make It Simple, Use Abstraction. URL:https://blog.testproject.io/2017/07/16/page-object-model/
5. Page Object Model (POM). URL: https://www.geeksforgeeks.org/page-object-model-pom/
6. Graham D., Black R., Erik van Veenendal. Foundations of Software Testing: ISTQB Certification, 4 th Edition. United Kingdom : EMEA, 2018. 273 p.
7. Seretta Gamba, Dorothy Graham. A Journey through Test Automation Patterns: One team's adventures with the Test Automation Patterns. United States: CreateSpace Independent Publishing Platform, 2018. 364 p.
8. Karl Wiegers, Joy Beatty. Software Requirements: Third Edition. Washington : Microsoft Press, 2013. 673 p.
9. Acceptance Criteria for User Stories: Purposes, Formats, Examples, and Best Practices. URL: https://www.altexsoft.com/blog/business/acceptance-criteria-purposes-formats-and-best-practices/
10. Maven репозиторій. URL: https://mvnrepository.com/

USING PROGRAMMING PATTERNS IN AUTOMATED TESTING

Harasymiv V. M.
Candidate of Technical Sciences, Associate Professor
Ivano-Frankivsk National Technical University of Oil and Gas
76019, 15 Karpatska St., Ivano-Frankivsk, Ukraine
https://orcid.org/0000-0002-6613-3549
e-mail: vira.harasymiv@nung.edu.ua

Abstract. The article discusses the features of building projects that implement the Page Object and Page Factory design patterns for creating automated tests. The main programming language chosen was Java, which belongs to the object-oriented paradigm, and the projects were developed in the IntelliJ IDEA environment. The focus is on testing the process of registering a new user on a web resource. The corresponding autotests were implemented using the specified design patterns together with the Selenium-java, WebDriverManager libraries and the TestNG test framework. The data required for successful registration is read from the configuration file in .properties format, which is stored in the resources directory. The result of the test is a generated report that demonstrates the success of the autotest. Each web page is represented by a separate class, in which methods for interacting with it are implemented. In the case of using only Page Object, the interface elements are hidden (have the private modifier) and placed in a separate module. Page methods return new instances of classes that correspond to subsequent stages of the test scenario. The autotest code is logically separated from interface elements and methods that simulate user actions. Also, thanks to the use of annotations @BeforeSuite, @AfterSuite and @BeforeClass, methods responsible for preparing and completing the execution of test suites are created. A comparison of project structures built using the Page Object and Page Factory templates is presented. Since Page Factory is a more optimized approach to initializing page elements or a page object, it allows you to make the project structure simpler. In this case, interface elements are placed directly in the corresponding page classes.

Keywords: project, design template, Page Object, Page Factory, autotest, interface elements, web application, test framework.