МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп’ютерної інженерії

ПАНАСЮК Іван Сергійович

# Програмний додаток інтеграції тестування на основі даних у фреймворк автоматизації/ Data-driven testing integration software application into an automation framework

спеціальність: 123 – Комп’ютерна інженерія
освітньо-професійна програма – Комп’ютерна інженерія

Кваліфікаційна робота

Виконав: студент групи КІ-41
Панасюк Іван Сергійович

Науковий керівник
к.т.н. Дубчак Л.О.

ТЕРНОПІЛЬ - 2026

# АНОТАЦІЯ

Панасюк І.С. Програмний додаток інтеграції тестування на основі даних у фреймворк автоматизації. – Рукопис.

Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр» за спеціальністю 123 «Комп'ютерна інженерія», освітньо-професійна програма. Західноукраїнський національний університет, Тернопіль, 2026.

Робота написана обсягом 71 сторінок і містить 18 рисунків, 3 таблиці та 4 додатки. Обсяг графічного матеріалу 2 аркуші формату А3.

Мета роботи полягає у розробці програмного додатка інтеграції тестування на основі даних у фреймворк автоматизації для підвищення ефективності виконання автоматизованих тестів та керування тестовими даними.

Методи дослідження включають аналіз науково-технічної літератури, порівняльний аналіз фреймворків автоматизації тестування, методи об'єктно-орієнтованого проєктування програмного забезпечення та експериментальне тестування розробленого програмного рішення.

У роботі проаналізовано методи автоматизації тестування програмного забезпечення, сучасні фреймворки автоматизації та особливості застосування підходу Data-Driven Testing. Виконано проєктування архітектури програмного додатка та розроблено механізм інтеграції тестування на основі даних у фреймворк автоматизації.

Результатом роботи є створений програмний додаток, який забезпечує автоматичне зчитування тестових даних із JSON-файлів, автоматизований запуск тестів за допомогою Jenkins та формування звітності засобами Allure Framework. Використання розробленого рішення дозволяє підвищити ефективність автоматизованого тестування та спростити підтримку тестового фреймворку.

Ключові слова: АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ, DATA-DRIVEN TESTING, ФРЕЙМВОРК АВТОМАТИЗАЦІЇ, JAVA, JSON, SELENIUM, TESTNG, JENKINS, ALLURE FRAMEWORK.

# ANNOTATION

Panasiuk I.S. Data-driven testing integration software application into an automation framework. – Manuscript.

Qualification work for the Bachelor degree in specialty 123 «Computer Engineering», educational and professional program. Western Ukrainian National University, Ternopil, 2026.

The thesis consists of 71 pages and contains 18 figures, 3 tables, and 4 appendices. The graphical material comprises 2 A3-format sheets.

The purpose of the thesis is to develop a software application for integrating data-driven testing into an automation framework in order to improve the efficiency of automated test execution and test data management.

The research methods include the analysis of scientific and technical literature, comparative analysis of test automation frameworks, object-oriented software design methods, and experimental testing of the developed software solution.

The thesis analyzes software test automation methods, modern test automation frameworks, and the specific features of applying the Data-Driven Testing approach. The architecture of the software application was designed, and a mechanism for integrating data-driven testing into an automation framework was developed.

The result of the work is a software application that provides automatic reading of test data from JSON files, automated test execution using Jenkins, and report generation through the Allure Framework. The developed solution improves the efficiency of automated testing and simplifies the maintenance of the test automation framework.

Keywords: AUTOMATED TESTING, DATA-DRIVEN TESTING, TEST AUTOMATION FRAMEWORK, JAVA, JSON, SELENIUM, TESTNG, JENKINS, ALLURE FRAMEWORK.

ЗМІСТ

# ВСТУП

На сьогоднішній день розвиток інформаційних технологій забезпечення якості програмного забезпечення є одним із найважливіших етапів життєвого циклу розробки програмних продуктів. Зі збільшенням складності програмних систем, кількості функціональних можливостей та інтеграцій між компонентами виникає необхідність у швидкому, стабільному та ефективному процесі тестування. Саме тому автоматизоване тестування набуває все більшої актуальності, оскільки дозволяє значно скоротити час перевірки програмного забезпечення, підвищити точність результатів та зменшити вплив людського фактору.

Одним із перспективних напрямів розвитку автоматизованого тестування є використання підходу Data-Driven Testing, який базується на відокремленні тестових даних від логіки тестових сценаріїв. Такий підхід дозволяє виконувати один і той самий тест із різними наборами даних, що забезпечує масштабованість, повторне використання тестових сценаріїв та спрощення супроводу тестового фреймворку. Інтеграція тестування на основі даних у сучасні фреймворки автоматизації сприяє підвищенню гнучкості тестових систем та ефективності процесу забезпечення якості програмного забезпечення.

Актуальність теми випускної кваліфікаційної роботи полягає у необхідності створення програмного додатка, який забезпечить інтеграцію механізмів тестування на основі даних у фреймворк автоматизації з метою підвищення ефективності виконання тестів, спрощення роботи з тестовими даними та зменшення витрат часу на підтримку тестової інфраструктури. Використання такого програмного рішення дозволяє оптимізувати процес створення та виконання тестових сценаріїв у системах автоматизації тестування.

Метою кваліфікаційної роботи є розробка програмного додатка інтеграції тестування на основі даних у фреймворк автоматизації для підвищення

ефективності виконання автоматизованих тестів та централізованого керування тестовими даними.

Для досягнення поставленої мети необхідно виконати такі завдання:

- проаналізувати особливості автоматизованого тестування програмного забезпечення;
- дослідити сучасні фреймворки автоматизації тестування;
- проаналізувати принципи та особливості підходу Data-Driven Testing;
- визначити вимоги до програмного додатка інтеграції тестування;
- спроєктувати архітектуру програмного рішення;
- обрати технології та засоби реалізації програмного додатка;
- реалізувати програмний додаток інтеграції тестування на основі даних;
- провести тестування та оцінити ефективність розробленого рішення.

Об’єктом дослідження є процес автоматизованого тестування програмного забезпечення. Предметом дослідження є методи та засоби інтеграції тестування на основі даних у фреймворки автоматизації.

Результати проведеного дослідження та практичної реалізації програмного модуля були опубліковані на [1].

Структурно бакалаврська робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків [2–6].

У першому розділі розглянуто теоретичні основи автоматизованого тестування та підходу тестування на основі даних (Data-Driven Testing). У другому розділі виконано проєктування програмного додатка інтеграції тестування. У третьому розділі представлено реалізацію програмного рішення, результати тестування та оцінку ефективності його використання.

# 1 ОСОБЛИВОСТІ ТЕСТУВАННЯ НА ОСНОВІ ДАНИХ У ФРЕЙМВОРКУ АВТОМАТИЗАЦІЇ

## 1.1 Характеристика основних процесів автоматизованого тестування програмного забезпечення

У процесі стрімкого розвитку інформаційних технологій програмне забезпечення використовується практично в усіх сферах людської діяльності, включаючи фінансовий сектор, медицину, промисловість, транспорт, освіту та державне управління [7]. Зростання складності програмних систем, збільшення кількості користувачів та необхідність постійного оновлення функціональних можливостей програмних продуктів висувають високі вимоги до забезпечення їх якості, стабільності та надійності [8]. Одним із ключових етапів життєвого циклу програмного забезпечення є тестування, яке дозволяє виявляти дефекти, перевіряти відповідність програмного продукту встановленим вимогам та забезпечувати його коректне функціонування в різних умовах експлуатації [9]. У зв’язку з цим автоматизоване тестування стало важливою складовою процесу розробки програмного забезпечення, оскільки дозволяє значно підвищити ефективність перевірки програмних систем та оптимізувати процес забезпечення якості [10].

Автоматизоване тестування являє собою процес використання спеціалізованих програмних засобів, скриптів та фреймворків для автоматичного виконання тестових сценаріїв без безпосереднього втручання користувача [11]. Основною метою такого підходу є скорочення часу на виконання повторюваних перевірок, зменшення кількості помилок, пов’язаних із людським фактором, а також забезпечення стабільності результатів тестування. На відміну від ручного тестування, де перевірка функціональності здійснюється тестувальником вручну, автоматизація дозволяє виконувати тести багаторазово, швидко та з однаковою точністю, що є особливо важливим у великих програмних проєктах із постійними змінами програмного коду [12].

Розробка програмного забезпечення сьогодні базується на принципах безперервної інтеграції та безперервного розгортання, що вимагає регулярного виконання великої кількості тестів після кожної зміни програмного коду [13]. У таких умовах ручне тестування стає малоефективним через значні часові витрати та складність підтримки стабільності процесу перевірки. Саме тому автоматизоване тестування використовується як один із основних інструментів контролю якості програмного забезпечення. Його застосування дозволяє інтегрувати процес перевірки в автоматизовані конвеєри збірки та доставки програмних продуктів, що забезпечує швидке виявлення дефектів та оперативне реагування на помилки [14].

Основу автоматизованого тестування складають тестові сценарії, які являють собою послідовність дій, необхідних для перевірки певної функціональності програмного забезпечення. Такі сценарії створюються у вигляді програмного коду з використанням спеціалізованих бібліотек та фреймворків автоматизації. Під час виконання тестів програмний додаток автоматично виконує необхідні дії, взаємодіє з інтерфейсом користувача, надсилає запити до сервера, перевіряє відповіді системи та формує результати тестування. Отримані результати порівнюються з очікуваними значеннями, після чого визначається успішність проходження тесту [15].

Однією з ключових особливостей автоматизованого тестування є можливість багаторазового використання тестових сценаріїв. Один раз створений тест може виконуватись необмежену кількість разів у різних середовищах та для різних конфігурацій програмного забезпечення. Це дозволяє суттєво зменшити витрати часу на повторне тестування функціональності після внесення змін до системи. Особливо важливим це є для регресійного тестування, яке спрямоване на перевірку того, що нові зміни в коді не порушили вже реалізовану функціональність [16]. У великих проєктах регресійне тестування може включати сотні або навіть тисячі тестових сценаріїв, тому виконання таких перевірок вручну є практично неможливим.

Важливим аспектом автоматизованого тестування є забезпечення стабільності та відтворюваності результатів. Використання автоматизованих сценаріїв дозволяє уникнути впливу суб'єктивних факторів, які можуть виникати під час ручного тестування. Автоматизовані тести виконуються за однаковими правилами та умовами, що забезпечує точність і повторюваність перевірок. Це особливо актуально для складних програмних систем, де навіть незначні зміни можуть впливати на взаємодію між окремими компонентами системи [17].

Процес автоматизованого тестування передбачає використання різноманітних технологій та інструментів, які забезпечують створення, виконання та аналіз тестів. Фреймворки автоматизації дозволяють працювати з вебзастосунками, мобільними платформами, API-сервісами, базами даних та іншими компонентами програмних систем. Для реалізації тестових сценаріїв широко використовуються мови програмування Java, Python, JavaScript та C#, а також спеціалізовані інструменти, такі як Selenium, Playwright, Cypress, TestNG, JUnit та REST Assured [18]. Вибір конкретних технологій залежить від типу програмного забезпечення, вимог проєкту та особливостей тестового середовища.

Суттєву роль у процесі автоматизованого тестування відіграє управління тестовими даними. Ефективність тестових сценаріїв значною мірою залежить від правильності формування вхідних даних та можливості їх повторного використання. У великих проєктах виникає необхідність тестування однієї функціональності з великою кількістю різних наборів даних, що призводить до ускладнення структури тестів та збільшення обсягу тестового коду. Для вирішення цієї проблеми застосовується підхід Data-Driven Testing, який передбачає відокремлення тестових даних від логіки тестових сценаріїв [19]. Це дозволяє виконувати один тест із різними наборами параметрів без дублювання коду, що значно спрощує підтримку та масштабування автоматизованого тестування.

Автоматизація тестування також сприяє покращенню процесу документування результатів перевірки програмного забезпечення. Більшість фреймворків підтримують генерацію автоматичних звітів, які містять інформацію

про успішність виконання тестів, виявлені помилки, час виконання сценаріїв та інші статистичні показники [20]. Такі звіти дозволяють швидко оцінити стан програмного продукту та виявити проблемні компоненти системи. Крім того, результати тестування можуть інтегруватися із системами моніторингу та управління проєктами, що забезпечує централізоване керування процесом забезпечення якості.

Важливим етапом автоматизованого тестування є інтеграція тестових процесів у середовище безперервної інтеграції. Системи розробки програмного забезпечення використовують спеціалізовані платформи, які автоматично запускають тести після внесення змін до програмного коду. Такий підхід дозволяє швидко виявляти дефекти на ранніх етапах розробки та запобігати накопиченню помилок у системі. Автоматичне виконання тестів після кожного оновлення коду забезпечує постійний контроль якості програмного забезпечення та сприяє стабільності процесу розробки [21].

Попри значні переваги автоматизованого тестування, його впровадження потребує ретельного планування та правильного вибору підходів до реалізації тестової інфраструктури. Створення автоматизованих тестів вимагає значних початкових витрат часу та ресурсів, пов’язаних із розробкою тестових сценаріїв, налаштуванням середовища та підтримкою тестового фреймворку. Крім того, автоматизовані тести потребують регулярного оновлення у разі зміни функціональності програмного забезпечення або модифікації інтерфейсу користувача. Неправильна організація тестового коду може призвести до зниження ефективності автоматизації та ускладнення підтримки тестових сценаріїв.

Однією з актуальних проблем автоматизованого тестування є забезпечення гнучкості та масштабованості тестових систем. У процесі розвитку програмного продукту кількість тестових сценаріїв постійно зростає, що призводить до збільшення складності управління тестами та тестовими даними. Для вирішення цієї проблеми використовуються різноманітні архітектурні підходи та шаблони проєктування, які дозволяють організувати структуру тестового фреймворку та

забезпечити повторне використання програмних компонентів. Одним із найбільш ефективних підходів є використання модульної архітектури та принципів розділення відповідальності між компонентами системи [22].

Таким чином, автоматизоване тестування є невід'ємною складовою процесу розробки програмного забезпечення, яка забезпечує підвищення якості програмних продуктів, скорочення часу тестування та оптимізацію процесів контролю якості. Використання автоматизації дозволяє виконувати складні та багаторазові перевірки з високою точністю, забезпечувати стабільність результатів та інтегрувати процес тестування у системи безперервної розробки. Подальший розвиток автоматизованого тестування пов'язаний із впровадженням нових технологій, підвищенням рівня масштабованості тестових систем та вдосконаленням механізмів управління тестовими даними, що є особливо важливим для реалізації підходу тестування на основі даних у фреймворках автоматизації.

## 1.2 Аналіз сучасних фреймворків автоматизації тестування

Стрімкий розвиток інформаційних технологій та постійне зростання складності програмних систем сприяли активному розвитку засобів автоматизації тестування програмного забезпечення. У процесі створення програмних продуктів особливого значення набуває забезпечення стабільності, надійності та якості програмного коду, що неможливо реалізувати без використання ефективних фреймворків автоматизації тестування [23]. Використання таких фреймворків дозволяє оптимізувати процес створення тестових сценаріїв, забезпечити повторне використання тестового коду, інтегрувати тестування у процес безперервної інтеграції та значно скоротити витрати часу на перевірку функціональності програмного забезпечення [24].

Фреймворк автоматизації тестування являє собою комплекс програмних компонентів, бібліотек, інструментів та правил організації тестового процесу, які забезпечують створення, виконання та підтримку автоматизованих тестів. Основною метою використання фреймворків є спрощення процесу автоматизації, стандартизація структури тестового коду та підвищення ефективності тестування [25]. Завдяки використанню фреймворків забезпечується можливість централізованого керування тестовими сценаріями, формування звітності, інтеграції з системами контролю версій та підтримки різних типів тестування.

Одним із найбільш поширених фреймворків автоматизації тестування вебзастосунків є Selenium. Даний інструмент використовується для автоматизації взаємодії з веббраузерами та підтримує більшість мов програмування, включаючи Java, Python, C# та JavaScript [26]. Selenium дозволяє створювати автоматизовані сценарії для перевірки функціональності вебінтерфейсів, емуляції дій користувача та виконання регресійного тестування. Значною перевагою цього фреймворку є підтримка різних браузерів та можливість інтеграції з іншими інструментами автоматизації. Водночас Selenium потребує додаткового налаштування тестового середовища та може ускладнювати підтримку тестів у разі частих змін структури вебінтерфейсу.

Іншим популярним рішенням є Playwright, який останніми роками активно використовується у сфері автоматизації тестування вебзастосунків. Основною особливістю Playwright є висока швидкість виконання тестів, підтримка багатопотокового тестування та можливість автоматизації вебтехнологій [27]. Даний фреймворк підтримує роботу з браузерами Chromium, Firefox та WebKit, а також забезпечує вбудовані механізми очікування елементів, що підвищує стабільність виконання тестів. Завдяки інтеграції з JavaScript та TypeScript Playwright широко застосовується у frontend-проєктах.

Для автоматизації тестування API-сервісів широко використовуються фреймворки REST Assured та Postman. REST Assured дозволяє створювати автоматизовані тести для REST API із використанням мови Java та забезпечує

зручні механізми перевірки HTTP-запитів і відповідей сервера [28]. Використання цього інструменту дозволяє реалізовувати інтеграційне тестування та перевіряти коректність роботи серверної логіки програмного забезпечення. Postman, у свою чергу, забезпечує зручний графічний інтерфейс для створення та виконання API-запитів, що робить його ефективним інструментом як для ручного, так і для автоматизованого тестування.

У процесі автоматизації тестування важливу роль відіграють фреймворки управління виконанням тестів. Одними з найбільш популярних рішень є JUnit та TestNG. Дані фреймворки забезпечують запуск тестових сценаріїв, формування звітності та підтримку параметризованого тестування. TestNG додатково підтримує паралельне виконання тестів, залежності між тестовими сценаріями та розширені механізми конфігурації тестового середовища [29]. Використання таких інструментів дозволяє підвищити продуктивність тестування та оптимізувати процес виконання великої кількості автоматизованих перевірок.

Одним із важливих аспектів використання фреймворків автоматизації є підтримка підходу Data-Driven Testing. У проєктах тестові сценарії часто потребують виконання з великою кількістю різних наборів даних, що ускладнює структуру тестового коду та призводить до дублювання логіки тестування. Використання Data-Driven підходу дозволяє відокремити тестові дані від програмного коду тестів, забезпечуючи централізоване керування наборами даних та можливість повторного використання тестових сценаріїв. Більшість фреймворків підтримують інтеграцію з файлами CSV, Excel, JSON, XML та базами даних для зберігання тестових параметрів.

Суттєвою перевагою фреймворків автоматизації є можливість інтеграції з інструментами безперервної інтеграції та доставки програмного забезпечення. Такі системи, як Jenkins, GitLab CI/CD та GitHub Actions, дозволяють автоматично запускати тестові сценарії після кожної зміни програмного коду. Це забезпечує швидке виявлення дефектів та стабільність процесу розробки програмного забезпечення. Інтеграція тестових фреймворків із системами контролю версій та

платформами моніторингу дозволяє централізовано керувати процесом забезпечення якості та формувати детальну аналітику результатів тестування.

Для забезпечення ефективності автоматизованого тестування важливе значення має правильний вибір фреймворку автоматизації. Під час вибору необхідно враховувати тип програмного забезпечення, мову програмування, складність проєкту, потребу у підтримці паралельного виконання тестів, інтеграцію з CI/CD-системами та можливість роботи з тестовими даними. Крім того, важливим фактором є масштабованість фреймворку та простота підтримки тестового коду в умовах постійного розвитку програмного продукту.

Для порівняння основних характеристик фреймворків автоматизації тестування доцільно використати таблицю 1.1.

Таблиця 1.1 – Порівняльна характеристика фреймворків автотестування

| Фреймворк | Основне призначення | Мови програмування | Переваги | Недоліки |
|---|---|---|---|---|
| Selenium | Автоматизація вебтестування | Java, Python, C#, JavaScript | Підтримка багатьох браузерів, | Складність налаштування |
| Playwright | Автоматизація вебзастосунків | JavaScript, TypeScript, Java | Висока швидкість, стабільність тестів | Відносно новий фреймворк |
| REST Assured | Тестування REST API | Java | Зручна робота з HTTP-запитами | Орієнтація лише на API |
| Postman | Тестування API | JavaScript | Простий інтерфейс, швидке налаштування | Обмежена масштабованість |
| JUnit | Управління виконанням тестів | Java | Простота використання | Менше можливостей для конфігурації |
| TestNG | Організація тестових сценаріїв | Java | Паралельне виконання, параметризація | Складніша конфігурація |

Використання фреймворків автоматизації тестування є необхідною складовою процесу забезпечення якості програмного забезпечення. Сучасні інструменти автоматизації дозволяють оптимізувати процес створення тестових сценаріїв, інтегрувати тестування у процес безперервної розробки та забезпечити масштабованість тестових систем. Аналіз функціональних можливостей існуючих фреймворків свідчить про доцільність використання підходу Data-Driven Testing, який дозволяє підвищити ефективність автоматизованого тестування та спростити управління тестовими даними у програмних проєктах.

## 1.3 Обґрунтування вибору підходу тестування на основі даних у системах автоматизації та постановка задачі дослідження

У процесі розвитку програмного забезпечення суттєво зростає складність тестових сценаріїв, кількість функціональних перевірок та обсяг тестових даних, необхідних для забезпечення належного рівня якості програмних продуктів [30]. У великих програмних проєктах одна й та сама функціональність часто потребує перевірки з використанням значної кількості різних наборів вхідних параметрів, що призводить до дублювання тестового коду, збільшення часу підтримки тестових сценаріїв та ускладнення процесу автоматизації тестування. У зв’язку з цим особливого значення набуває використання підходів, спрямованих на оптимізацію структури тестового фреймворку та підвищення ефективності роботи з тестовими даними.

Одним із найбільш ефективних підходів у сфері автоматизованого тестування є Data-Driven Testing, який передбачає відокремлення тестових даних від логіки виконання тестових сценаріїв [31]. Основною ідеєю даного підходу є використання одного тестового сценарію для виконання перевірок із різними наборами параметрів, які зберігаються у зовнішніх джерелах даних. Це дозволяє

значно скоротити обсяг тестового коду, підвищити рівень повторного використання тестових сценаріїв та забезпечити централізоване керування тестовими даними.

Традиційний підхід до автоматизації тестування часто передбачає створення окремого тестового сценарію для кожного набору вхідних даних. У результаті зі збільшенням кількості тестових випадків суттєво зростає обсяг програмного коду, що негативно впливає на підтримку тестового фреймворку та ускладнює внесення змін до тестових сценаріїв. Використання Data-Driven Testing дозволяє уникнути дублювання коду шляхом параметризації тестів та передачі тестових даних із зовнішніх джерел, таких як CSV-файли, Excel-таблиці, JSON-документи, XML-файли або бази даних [32].

Важливою перевагою підходу тестування на основі даних є підвищення масштабованості автоматизованого тестування. Один тестовий сценарій може використовуватись для виконання великої кількості перевірок без необхідності створення нових тестів. Це особливо актуально для систем, у яких необхідно перевіряти роботу програмного забезпечення з різними наборами параметрів, ролями користувачів, типами даних або конфігураціями середовища. Використання параметризованих тестів дозволяє швидко розширювати тестове покриття та забезпечує ефективну підтримку тестового фреймворку в процесі розвитку програмного продукту.

Суттєвим аспектом використання Data-Driven Testing є централізоване керування тестовими даними. У традиційних системах автоматизації тестові дані часто безпосередньо інтегруються у програмний код тестів, що ускладнює їх оновлення та повторне використання. Відокремлення тестових даних від тестових сценаріїв дозволяє змінювати параметри тестування без модифікації логіки тестів, що значно спрощує підтримку автоматизованої системи тестування [33]. Крім того, централізоване зберігання тестових даних забезпечує можливість використання єдиних наборів параметрів для різних типів тестів та різних компонентів програмного забезпечення.

Використання підходу тестування на основі даних також сприяє покращенню організації тестового процесу. Завдяки чіткому розділенню логіки тестування та тестових даних підвищується читабельність тестового коду та спрощується взаємодія між учасниками процесу забезпечення якості. Тестувальники можуть працювати з тестовими даними незалежно від програмного коду, а розробники тестового фреймворку мають можливість зосередитись на реалізації механізмів виконання тестів. Такий підхід сприяє підвищенню продуктивності команди та зменшенню кількості помилок у тестових сценаріях.

Важливим фактором ефективності Data-Driven Testing є можливість інтеграції з теперішніми фреймворками автоматизації тестування. Більшість популярних інструментів, таких як Selenium, TestNG, JUnit, Playwright та REST Assured, підтримують механізми параметризації тестів та роботу із зовнішніми джерелами даних [34]. Це дозволяє реалізовувати масштабовані системи автоматизованого тестування, які можуть виконувати велику кількість тестових сценаріїв із різними наборами параметрів без значного збільшення складності тестового коду.

Однією з переваг підходу тестування на основі даних є підвищення ефективності регресійного тестування. У процесі оновлення програмного забезпечення виникає необхідність багаторазового виконання однакових тестових перевірок із різними наборами даних. Використання Data-Driven Testing дозволяє автоматизувати цей процес та забезпечити швидке виконання великої кількості тестових сценаріїв. Це суттєво скорочує час тестування та дозволяє оперативно виявляти дефекти після внесення змін до програмного коду.

Водночас впровадження підходу тестування на основі даних потребує правильного проєктування структури тестового фреймворку та організації роботи з тестовими даними. Неправильна реалізація механізмів параметризації може призвести до ускладнення тестового коду та зниження продуктивності системи автоматизації. Особливо важливим є забезпечення коректної структури зберігання тестових даних, механізмів їх обробки та підтримки повторного використання

тестових сценаріїв. Для ефективного використання Data-Driven Testing необхідно враховувати особливості архітектури програмного забезпечення, тип тестованої системи та вимоги до тестового покриття.

У системах автоматизації тестування підхід Data-Driven Testing часто поєднується з іншими методами організації тестового процесу, зокрема модульною архітектурою тестових фреймворків, шаблоном Page Object та інтеграцією із системами безперервної інтеграції. Використання таких підходів дозволяє створювати масштабовані та гнучкі системи автоматизації тестування, які забезпечують високу швидкість виконання тестів та простоту підтримки тестової інфраструктури [35].

Застосування тестування на основі даних є особливо актуальним для великих корпоративних систем, вебзастосунків, API-сервісів та програмних платформ із великою кількістю користувацьких сценаріїв. У таких системах використання Data-Driven Testing дозволяє значно підвищити рівень тестового покриття та забезпечити ефективне керування тестовими даними. Крім того, централізація роботи з тестовими параметрами сприяє стандартизації процесу автоматизованого тестування та підвищенню стабільності тестових сценаріїв.

Підхід тестування на основі даних є ефективним засобом оптимізації процесу автоматизованого тестування програмного забезпечення. Його використання дозволяє зменшити дублювання тестового коду, забезпечити масштабованість тестових систем, централізувати керування тестовими даними та підвищити ефективність регресійного тестування.

Проведений аналіз особливостей автоматизованого тестування, сучасних фреймворків автоматизації та підходу Data-Driven Testing дозволив встановити доцільність використання тестування на основі даних для підвищення ефективності процесу забезпечення якості програмного забезпечення. Встановлено, що використання зовнішніх джерел тестових даних сприяє зменшенню дублювання тестового коду, підвищенню масштабованості тестового фреймворку та спрощенню підтримки автоматизованих тестів.

На основі проведеного дослідження сформульовано задачу кваліфікаційної роботи, яка полягає у розробці програмного додатка інтеграції тестування на основі даних у фреймворк автоматизації з використанням мови програмування Java та формату JSON для зберігання тестових даних. Розроблюване програмне рішення повинно забезпечувати автоматичне зчитування та обробку тестових даних, передачу параметрів у тестові сценарії, підтримку автоматизованого запуску тестів за допомогою Jenkins та формування звітності про результати тестування засобами Allure Framework. Реалізація поставленої задачі має забезпечити підвищення ефективності автоматизованого тестування, спрощення роботи з тестовими даними та створення масштабованої системи автоматизації, придатної для подальшого розвитку та інтеграції в процес безперервної розробки програмного забезпечення.

# 2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ДОДАТКА ІНТЕГРАЦІЇ ТЕСТУВАННЯ НА ОСНОВІ ДАНИХ

## 2.1 Основні етапи розробки програмного додатка

Розробка програмного додатка інтеграції тестування на основі даних у фреймворк автоматизації є складним процесом, який включає аналіз вимог, проєктування архітектури, вибір технологій реалізації, створення механізмів роботи з тестовими даними та інтеграцію із засобами автоматизованого виконання тестів і формування звітності. У процесі створення програмного рішення особливу увагу приділено забезпеченню масштабованості системи, повторному використанню тестових сценаріїв та централізованому керуванню тестовими даними.

На початковому етапі розробки було виконано аналіз предметної області та визначено основні вимоги до програмного додатка. Основною метою системи є забезпечення інтеграції підходу Data-Driven Testing у фреймворк автоматизації тестування для підвищення ефективності виконання тестових сценаріїв та спрощення процесу роботи з тестовими даними. У результаті аналізу було встановлено, що програмний додаток повинен забезпечувати автоматичне зчитування тестових даних із JSON-файлів, передачу параметрів у тестові сценарії, підтримку запуску тестів через Jenkins та формування звітності за допомогою Allure Framework.

Однією з ключових вимог до програмного рішення стала реалізація централізованого механізму зберігання тестових даних. Для цього було обрано формат JSON, який забезпечує просту структуру зберігання інформації, підтримує вкладені об’єкти та дозволяє ефективно працювати з параметризованими тестами. Використання JSON-файлів дозволяє відокремити тестові дані від програмного коду тестових сценаріїв, що забезпечує гнучкість системи автоматизації та спрощує підтримку тестового фреймворку.

Після аналізу вимог було визначено основні етапи реалізації програмного додатка. Загальну схему процесу розробки представлено на рисунку 2.1.

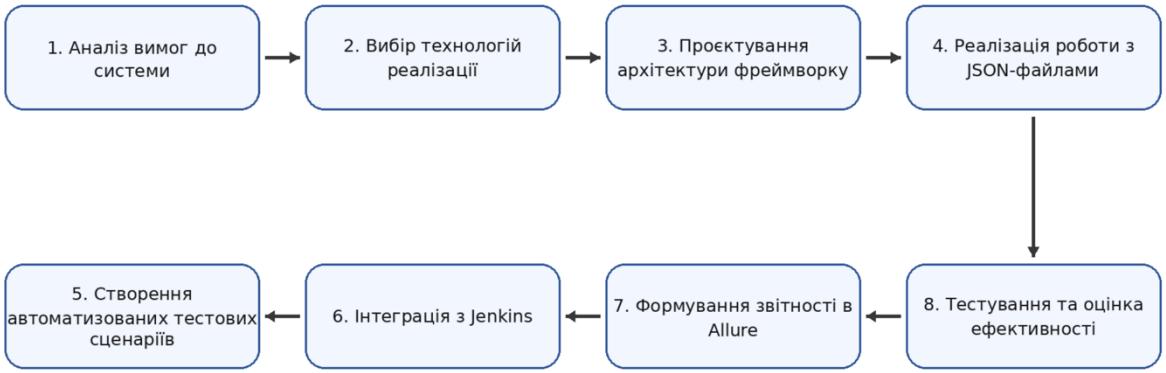


Рисунок 2.1 – Основні етапи розробки програмного додатка інтеграції тестування

Наступним етапом стало проєктування структури програмного додатка та визначення основних програмних модулів системи. Для реалізації програмного рішення було обрано мову програмування Java, оскільки вона забезпечує платформонезалежність, підтримку об’єктно-орієнтованого програмування та велику кількість бібліотек для автоматизації тестування. Крім того, Java широко використовується у сфері розробки автоматизованих тестових систем та підтримується більшістю фреймворків автоматизації.

У процесі проєктування було визначено структуру взаємодії між основними компонентами системи. Архітектура програмного додатка включає модуль роботи з тестовими даними, модуль виконання тестів, систему формування звітності та механізм інтеграції із сервером безперервної інтеграції Jenkins. Загальну структуру взаємодії компонентів системи наведено на рисунку 2.2.

Важливим етапом реалізації програмного додатка стало створення механізму зчитування тестових даних із JSON-файлів. Для цього у середовищі Java було реалізовано окремий модуль Data Reader, який забезпечує автоматичне зчитування параметрів тестування та передачу даних у тестові сценарії.

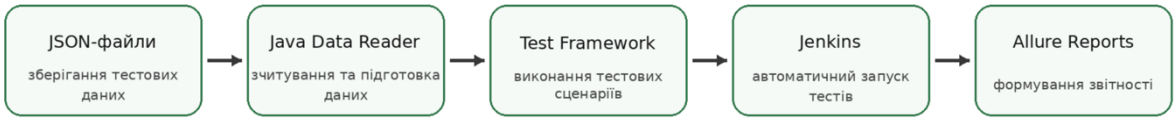


Рисунок 2.2 – Структура взаємодії компонентів програмного додатка

Використання такого підходу дозволяє запускати один тест із різними наборами даних без дублювання програмного коду. Структура JSON-файлу тестових даних представлена на рисунку 2.3.

test-data.json

```
{
  "testData": [
    {
      "testCaseId": "TC_001",
      "username": "admin",
      "password": "admin123",
      "expectedStatus": 200
    },
    {
      "testCaseId": "TC_002",
      "username": "user",
      "password": "user123",
      "expectedStatus": 200
    },
    {
      "testCaseId": "TC_003",
      "username": "invalid_user",
      "password": "wrong_password",
      "expectedStatus": 401
    }
```

UTF-8 JSON 4 spaces

Рисунок 2.3 – Приклад структури JSON-файлу тестових даних

Використання JSON-файлів забезпечує централізоване керування тестовими параметрами та спрощує масштабування системи автоматизації. У разі необхідності додавання нових тестових сценаріїв достатньо лише оновити набір тестових даних без зміни логіки програмного коду.

Після реалізації механізму роботи з тестовими даними було створено тестові сценарії із використанням Java та фреймворку автоматизації. Для організації

тестового процесу використовувався TestNG, який забезпечує підтримку параметризованого тестування, паралельного запуску тестів та інтеграцію з Allure Framework. Використання параметризованих тестів дозволило реалізувати Data-Driven підхід та забезпечити повторне використання тестових сценаріїв.

Суттєву роль у процесі автоматизації відіграє інтеграція із системою Jenkins, яка використовується для автоматичного запуску тестів після внесення змін до програмного коду. Jenkins дозволяє організувати процес безперервної інтеграції та забезпечує автоматичне виконання тестових сценаріїв у заданому середовищі. Після завершення виконання тестів система автоматично формує результати перевірки та передає їх до Allure Framework для створення звітності.

Схему інтеграції системи автоматизації з Jenkins та Allure представлено на рисунку 2.4.

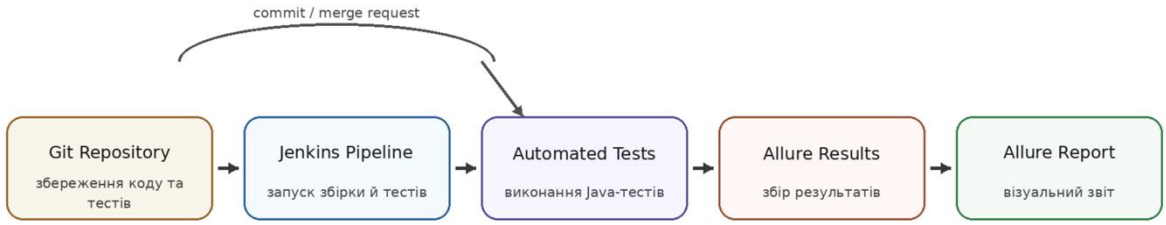


Рисунок 2.4 – Інтеграція тестового фреймворку з Jenkins та Allure

Використання Jenkins дозволяє автоматизувати процес запуску тестів, скоротити час перевірки програмного забезпечення та забезпечити постійний контроль якості програмного продукту. Крім того, Jenkins підтримує інтеграцію з GitHub та GitLab, що забезпечує автоматичний запуск тестів після кожного оновлення програмного коду.

Одним із важливих етапів реалізації програмного додатка стало впровадження системи формування звітності за допомогою Allure Framework. Даний інструмент забезпечує генерацію інтерактивних звітів про результати виконання тестів, відображення статистики проходження тестових сценаріїв, часу

виконання тестів та інформації про виявлені помилки. Використання Allure дозволяє підвищити зручність аналізу результатів тестування та спрощує процес моніторингу стану програмного забезпечення.

Після завершення реалізації програмного додатка було проведено тестування працездатності системи та оцінку ефективності використання підходу Data-Driven Testing. У результаті виконаного дослідження встановлено, що використання JSON-файлів для зберігання тестових даних дозволило зменшити дублювання тестового коду, спростити підтримку тестових сценаріїв та підвищити масштабованість системи автоматизації. Інтеграція з Jenkins забезпечила автоматизацію процесу запуску тестів, а використання Allure Framework дозволило реалізувати ефективну систему формування звітності.

Таким чином, у процесі розробки програмного додатка було реалізовано комплексне рішення для інтеграції тестування на основі даних у фреймворк автоматизації. Використання мови програмування Java, JSON-файлів, Jenkins та Allure Framework дозволило створити масштабовану систему автоматизованого тестування, яка забезпечує централізоване керування тестовими даними, автоматичне виконання тестових сценаріїв та формування детальної звітності про результати тестування.

## 2.2 Архітектура програмного додатка інтеграції тестування

Архітектура програмного додатка є одним із найважливіших етапів проєктування системи автоматизації тестування, оскільки саме від правильності побудови структури програмних компонентів залежить масштабованість, стабільність та ефективність роботи тестового фреймворку. У процесі розробки програмного додатка інтеграції тестування на основі даних було реалізовано модульну архітектуру, яка забезпечує розділення відповідальності між окремими

компонентами системи, спрощує підтримку тестового коду та дозволяє централізовано керувати тестовими даними.

Основною метою створення архітектури програмного додатка стало забезпечення інтеграції підходу Data-Driven Testing у фреймворк автоматизації тестування з використанням мови програмування Java, JSON-файлів, Jenkins та Allure Framework. Архітектура системи повинна забезпечувати автоматичне зчитування тестових даних, передачу параметрів у тестові сценарії, запуск тестів у середовищі безперервної інтеграції та формування детальної звітності про результати виконання тестів.

На початковому етапі проєктування було визначено основні компоненти програмного додатка та механізми їх взаємодії. До складу системи входять модуль зберігання тестових даних, модуль обробки JSON-файлів, модуль виконання тестових сценаріїв, модуль формування звітності та сервер автоматизованого запуску тестів Jenkins.

Основою архітектури програмного додатка є модуль роботи з тестовими даними. Для реалізації підходу Data-Driven Testing тестові параметри зберігаються у зовнішніх JSON-файлах, які містять необхідні вхідні значення для виконання тестових сценаріїв. Використання JSON забезпечує зручний формат зберігання інформації, підтримує вкладені структури даних та дозволяє легко масштабувати систему шляхом додавання нових тестових наборів.

Для обробки JSON-файлів у середовищі Java реалізовано окремий програмний компонент Data Reader, який виконує зчитування тестових даних та передачу параметрів у тестові сценарії. Даний модуль забезпечує централізовану роботу з тестовими даними та дозволяє повторно використовувати тестові набори у різних тестових сценаріях. Крім того, використання окремого Data Reader дозволяє змінювати формат зберігання тестових даних без модифікації логіки тестових сценаріїв.

Схему взаємодії модуля роботи з JSON-файлами представлено на рисунку 2.5.

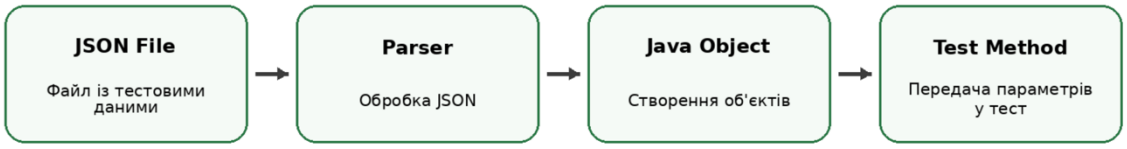


Рисунок 2.5 – Схема роботи модуля зчитування тестових даних

Після зчитування тестових даних система передає параметри до тестового шару, який реалізований із використанням Java та TestNG. У структурі тестового фреймворку кожен тестовий сценарій отримує необхідні параметри автоматично, що дозволяє виконувати один тест із різними наборами даних без дублювання коду. Такий підхід забезпечує масштабованість системи автоматизації та значно спрощує підтримку тестових сценаріїв.

Важливим компонентом архітектури є модуль автоматизованого запуску тестів, реалізований із використанням Jenkins. Даний сервер безперервної інтеграції забезпечує автоматичний запуск тестових сценаріїв після внесення змін до програмного коду або за заданим розкладом. Jenkins інтегрується з Git-репозиторієм, виконує збірку проєкту, запускає автоматизовані тести та формує результати тестування. Такий підхід дозволяє забезпечити постійний контроль якості програмного забезпечення та автоматизувати процес перевірки функціональності системи.

Схему процесу автоматичного запуску тестів у Jenkins наведено на рисунку 2.6.

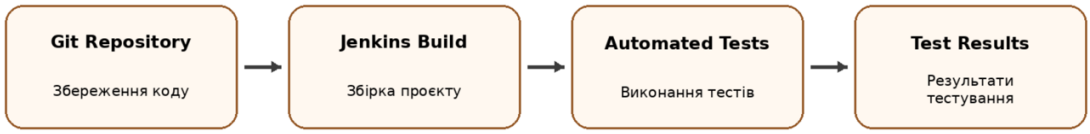


Рисунок 2.6 – Процес автоматизованого запуску тестів у Jenkins

Після завершення виконання тестів результати передаються до системи формування звітності Allure Framework. Використання Allure забезпечує генерацію інтерактивних звітів, які містять інформацію про кількість успішних та неуспішних тестів, тривалість виконання тестових сценаріїв, помилки та статистику проходження тестів. Інтеграція Allure з TestNG та Jenkins дозволяє автоматично формувати звіти після кожного запуску тестового набору.

У процесі проєктування архітектури програмного додатка особливу увагу приділено принципам модульності та повторного використання компонентів. Структура тестового фреймворку побудована таким чином, щоб кожен модуль виконував окрему функцію та міг бути змінений незалежно від інших компонентів системи. Такий підхід дозволяє спростити підтримку програмного коду та забезпечує можливість масштабування системи автоматизації у майбутньому.

Архітектура програмного додатка також передбачає використання допоміжних програмних компонентів для роботи з конфігураційними файлами, логуванням та обробкою помилок. Для зберігання конфігурацій використовуються окремі файли налаштувань, які містять параметри запуску тестів, URL-адреси тестових середовищ та дані для підключення до системи. Логування процесу виконання тестів дозволяє відслідковувати помилки та аналізувати результати роботи системи автоматизації.

Важливою особливістю архітектури є підтримка масштабованості тестового фреймворку. У разі необхідності система дозволяє додавати нові тестові сценарії, модулі роботи з даними або інтеграцію з іншими сервісами без значної зміни існуючої структури програмного додатка. Завдяки використанню модульного підходу та відокремленню тестових даних від логіки тестування забезпечується висока гнучкість системи автоматизації.

Крім того, реалізована архітектура забезпечує підтримку повторного використання тестових компонентів. Наприклад, один і той самий Data Reader може використовуватись для роботи з різними наборами тестових даних, а тестові методи можуть виконуватись із великою кількістю параметрів без створення нових

сценаріїв. Це дозволяє значно скоротити обсяг тестового коду та підвищити ефективність підтримки тестового фреймворку.

Таким чином, у процесі проєктування архітектури програмного додатка було створено модульну систему інтеграції тестування на основі даних, яка забезпечує централізоване керування тестовими даними, автоматичне виконання тестових сценаріїв та формування звітності про результати тестування. Використання Java, JSON-файлів, Jenkins та Allure Framework дозволило створити масштабовану та гнучку систему автоматизації тестування, яка забезпечує ефективну інтеграцію підходу Data-Driven Testing у процес забезпечення якості програмного забезпечення.

## 2.3 Вибір засобів та технологій реалізації програмного рішення

У процесі розробки програмного додатка інтеграції тестування на основі даних важливе значення має правильний вибір засобів та технологій реалізації системи автоматизації. Від обраних інструментів залежить ефективність виконання тестових сценаріїв, масштабованість програмного рішення, швидкість обробки тестових даних та можливість інтеграції із системами безперервної інтеграції та звітності. Під час вибору технологічного стеку враховувалися вимоги до підтримки підходу Data-Driven Testing, можливість роботи з JSON-файлами, інтеграція з Jenkins та формування звітності за допомогою Allure Framework.

Основною мовою програмування для реалізації програмного додатка було обрано Java. Використання Java обумовлено її платформонезалежністю, широкими можливостями об’єктно-орієнтованого програмування та значною кількістю бібліотек для автоматизації тестування. Крім того, Java є однією з найпоширеніших мов у сфері розробки тестових фреймворків, що забезпечує підтримку великої кількості інструментів автоматизації та систем інтеграції [23].

Суттєвою перевагою Java є підтримка багатопотокового виконання тестів, що дозволяє оптимізувати час виконання тестових сценаріїв у великих програмних проєктах. Також Java забезпечує зручну інтеграцію з бібліотеками для роботи з JSON-структурами, REST API, системами логування та фреймворками автоматизованого тестування. Для реалізації програмного рішення використовувалося середовище розробки IntelliJ IDEA, яке забезпечує підтримку роботи з Maven, автоматичне керування залежностями та зручні інструменти для написання та налагодження тестового коду.

Для реалізації підходу Data-Driven Testing було обрано формат JSON для зберігання тестових даних. Використання JSON-файлів дозволяє відокремити тестові параметри від логіки тестових сценаріїв та забезпечити централізоване керування тестовими даними. Формат JSON є простим для читання, підтримує вкладені структури даних та забезпечує швидку обробку інформації у середовищі Java. Крім того, JSON широко використовується у вебсистемах та API-сервісах, що спрощує інтеграцію тестового фреймворку з різними компонентами програмного забезпечення.

Для роботи з JSON-файлами використовувалася бібліотека Jackson, яка забезпечує серіалізацію та десеріалізацію даних у Java-об’єкти. Використання Jackson дозволяє автоматично перетворювати структури JSON у програмні класи Java та спрощує процес передачі тестових параметрів у тестові сценарії. Такий підхід забезпечує ефективну роботу з тестовими даними та дозволяє легко масштабувати систему автоматизації тестування.

У процесі реалізації автоматизованих тестових сценаріїв використовувався фреймворк TestNG. Даний інструмент забезпечує підтримку параметризованого тестування, паралельного запуску тестів та інтеграцію з системами формування звітності. Використання TestNG дозволило реалізувати Data-Driven підхід та забезпечити автоматичну передачу параметрів із JSON-файлів у тестові методи. Крім того, TestNG підтримує групування тестів, налаштування порядку виконання тестових сценаріїв та конфігурацію тестового середовища.

Для автоматизації тестування вебзастосунків використовувався Selenium WebDriver, який забезпечує можливість взаємодії з веббраузерами та автоматичного виконання користувацьких сценаріїв. Selenium дозволяє емулятувати дії користувача, перевіряти роботу елементів інтерфейсу та реалізовувати регресійне тестування програмного забезпечення. Інтеграція Selenium із Java та TestNG забезпечує створення масштабованого тестового фреймворку та підтримку параметризованих тестів.

Важливим компонентом технологічного стеку є система безперервної інтеграції Jenkins. Даний інструмент використовується для автоматичного запуску тестових сценаріїв після внесення змін до програмного коду або за визначеним розкладом. Jenkins забезпечує автоматизацію процесу збірки проєкту, запуск тестів та формування результатів тестування. Інтеграція Jenkins із Git-репозиторієм дозволяє реалізувати механізм безперервної інтеграції та забезпечує постійний контроль якості програмного забезпечення.

Схему взаємодії основних технологій програмного додатка наведено на рисунку 2.7.


Рисунок 2.7 – Взаємодія технологій програмного додатка

Для формування звітності про результати тестування використовувався Allure Framework. Даний інструмент забезпечує створення інтерактивних звітів, які містять інформацію про проходження тестів, статистику виконання сценаріїв, помилки та час виконання тестів. Використання Allure дозволяє підвищити зручність аналізу результатів тестування та забезпечує централізоване представлення інформації про стан програмного продукту.

У процесі проєктування програмного додатка також використовувалися UML-діаграми, які дозволяють графічно представити структуру системи та взаємодію між її компонентами. Використання UML забезпечує спрощення процесу проєктування архітектури програмного рішення та дозволяє формалізувати структуру програмного додатка. Для опису взаємодії між основними модулями системи було використано UML-діаграму компонентів, яка відображає зв'язки між модулями роботи з тестовими даними, тестовим фреймворком, Jenkins та системою формування звітності.

Використання UML-діаграм дозволяє покращити розуміння архітектури програмного рішення, спростити документування структури системи та забезпечити можливість подальшого масштабування тестового фреймворку. Крім того, UML-діаграми використовуються для моделювання процесів взаємодії між компонентами системи та оптимізації структури програмного коду.

Важливим етапом вибору технологій реалізації стало забезпечення сумісності всіх компонентів програмного додатка. Використання Java, Selenium, TestNG, Jenkins та Allure забезпечує створення єдиного середовища автоматизації тестування, у якому всі компоненти можуть ефективно взаємодіяти між собою. Крім того, обраний технологічний стек дозволяє забезпечити підтримку масштабованості системи та можливість інтеграції нових компонентів у майбутньому.

У результаті проведеного аналізу було встановлено, що використання мови програмування Java, JSON-файлів, Selenium WebDriver, TestNG, Jenkins та Allure Framework є доцільним для реалізації програмного додатка інтеграції тестування на основі даних. Використання даних технологій забезпечує автоматизацію процесу виконання тестів, централізоване керування тестовими даними, інтеграцію із системами безперервної інтеграції та формування детальної звітності про результати тестування.

Таким чином, вибір засобів та технологій реалізації програмного рішення дозволив створити масштабований та гнучкий фреймворк автоматизації

тестування, який підтримує підхід Data-Driven Testing та забезпечує ефективне керування тестовими сценаріями й тестовими даними у процесі забезпечення якості програмного забезпечення.

# 3 РЕАЛІЗАЦІЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ ПРОГРАМНОГО МОДУЛЯ ІНТЕГРАЦІЇ ТЕСТУВАННЯ НА ОСНОВІ ДАНИХ

## 3.1 Реалізація програмного додатка інтеграції тестування на основі даних

Реалізація програмного додатка інтеграції тестування на основі даних є одним із ключових етапів створення системи автоматизації тестування. Основною метою реалізації стало створення масштабованого тестового фреймворку, який забезпечує автоматичне зчитування тестових даних із JSON-файлів, передачу параметрів у тестові сценарії, автоматизований запуск тестів та формування звітності про результати виконання тестування. Для реалізації програмного рішення використовувалися мова програмування Java, фреймворки Selenium і TestNG, система безперервної інтеграції Jenkins та інструмент формування звітності Allure Framework.

На початковому етапі реалізації було створено структуру програмного проєкту в середовищі IntelliJ IDEA. Організація проєкту базувалася на принципах модульної архітектури та розділення відповідальності між компонентами системи. Структура проєкту включає окремі пакети для тестових сценаріїв, модулів роботи з тестовими даними, конфігураційних файлів, утиліт та механізмів формування звітності. Загальну структуру програмного проєкту наведено на рисунку 3.1.

Основою реалізації програмного додатка став механізм роботи з тестовими даними у форматі JSON. Для цього було створено окремий клас DataReader, який забезпечує автоматичне зчитування параметрів тестування та перетворення JSON-структур у Java-об’єкти. Використання окремого модуля для роботи з тестовими даними дозволило відокремити логіку тестування від тестових параметрів та забезпечити повторне використання тестових сценаріїв.

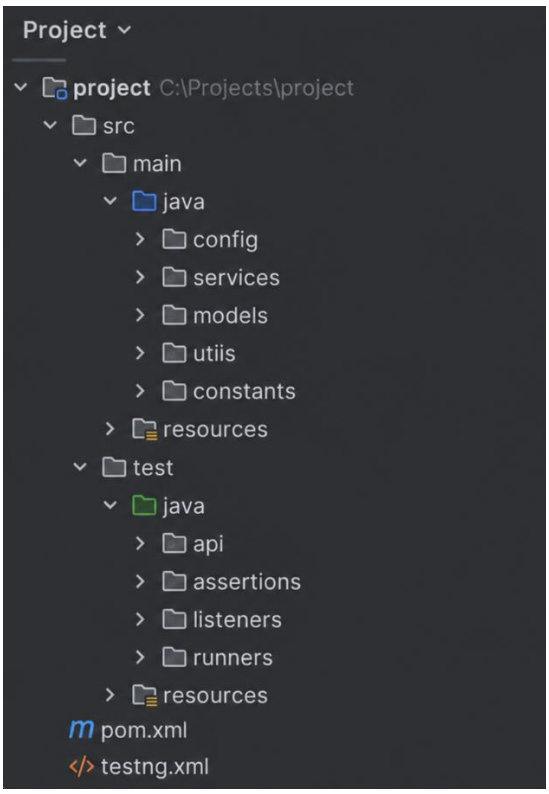


Рисунок 3.1 – Структура програмного проєкту в IntelliJ IDEA

Приклад реалізації класу зчитування тестових даних наведено на рисунку 3.2.

```
public class DataReader {

    public static List<UserData> getTestData() throws IOException {
        ObjectMapper mapper = new ObjectMapper();

        return Arrays.asList(
            mapper.readValue(
                new File("src/test/resources/testdata/users.json"),
                UserData[].class
            )
        );
    }
}
```

Рисунок 3.2 – Реалізація класу DataReader для роботи з JSON-файлами

Під час реалізації було створено окремий клас моделі даних UserData, який використовується для зберігання параметрів тестування. Використання Java-

об’єктів дозволяє спростити передачу даних між компонентами системи та забезпечує зручну роботу з параметризованими тестами.

Наступним етапом реалізації стало створення автоматизованих тестових сценаріїв із використанням Selenium WebDriver та TestNG. Для забезпечення підтримки Data-Driven Testing використовувався механізм параметризації тестів, який дозволяє автоматично передавати тестові дані у тестові методи.

Приклад реалізації параметризованого тесту наведено на рисунку 3.3.

```
@Test(dataProvider = "userData")
public void loginTest(UserData userData) {

    loginPage.enterUsername(userData.getUsername());
    loginPage.enterPassword(userData.getPassword());
    loginPage.clickLoginButton();

    Assert.assertTrue(homePage.isOpened());
}
```

Рисунок 3.3 – Реалізація параметризованого тесту

Для організації передачі тестових даних у тестові сценарії було реалізовано DataProvider TestNG. Даний механізм дозволяє автоматично передавати параметри з JSON-файлів у тестові методи та виконувати один тест із різними наборами даних.

У процесі реалізації тестового фреймворку використовувався шаблон проєктування Page Object Model. Даний підхід передбачає створення окремих класів для роботи з вебсторінками та дозволяє відокремити логіку взаємодії з елементами інтерфейсу від тестових сценаріїв. Використання Page Object Model забезпечує покращення структури тестового коду та спрощує підтримку автоматизованих тестів.

Приклад структури Page Object наведено на рисунку 3.4.

```
public class LoginPage {

    @FindBy(id = "username")
    private WebElement usernameField;

    @FindBy(id = "password")
    private WebElement passwordField;

    @FindBy(id = "loginButton")
    private WebElement loginButton;

    public void enterUsername(String username) {
        usernameField.sendKeys(username);
    }

    public void enterPassword(String password) {
        passwordField.sendKeys(password);
    }

    public void clickLoginButton() {
        loginButton.click();
    }
}
```

Рисунок 3.4 – Реалізація Page Object для сторінки авторизації

Для автоматизації процесу запуску тестів використовувалася система Jenkins. У процесі реалізації було створено Jenkins Pipeline, який забезпечує автоматичне виконання тестових сценаріїв після оновлення програмного коду в Git-репозиторії. Jenkins виконує збірку Maven-проєкту, запускає тестові сценарії та передає результати тестування до Allure Framework.

Для керування залежностями проєкту використовувався Maven. У файлі pom.xml були підключені бібліотеки Selenium, TestNG, Jackson та Allure Framework. Використання Maven дозволяє централізовано керувати залежностями проєкту та автоматизувати процес збірки програмного рішення. Фрагмент конфігурації Maven наведено на рисунку 3.5.

Важливим етапом реалізації програмного додатка стало впровадження системи формування звітності Allure Framework. Даний інструмент забезпечує генерацію інтерактивних звітів, які містять статистику проходження тестів, список успішних та неуспішних тестових сценаріїв, час виконання тестів та інформацію про помилки.

```xml
<dependencies>

    <dependency>
        <groupId>org.seleniumhq.selenium</groupId>
        <artifactId>selenium-java</artifactId>
        <version>4.20.0</version>
    </dependency>

    <dependency>
        <groupId>org.testng</groupId>
        <artifactId>testng</artifactId>
        <version>7.10.2</version>
    </dependency>

    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.17.0</version>
    </dependency>

</dependencies>
```

Рисунок 3.5 – Підключення залежностей у pom.xml

Приклад звіту Allure наведено на рисунку 3.6.

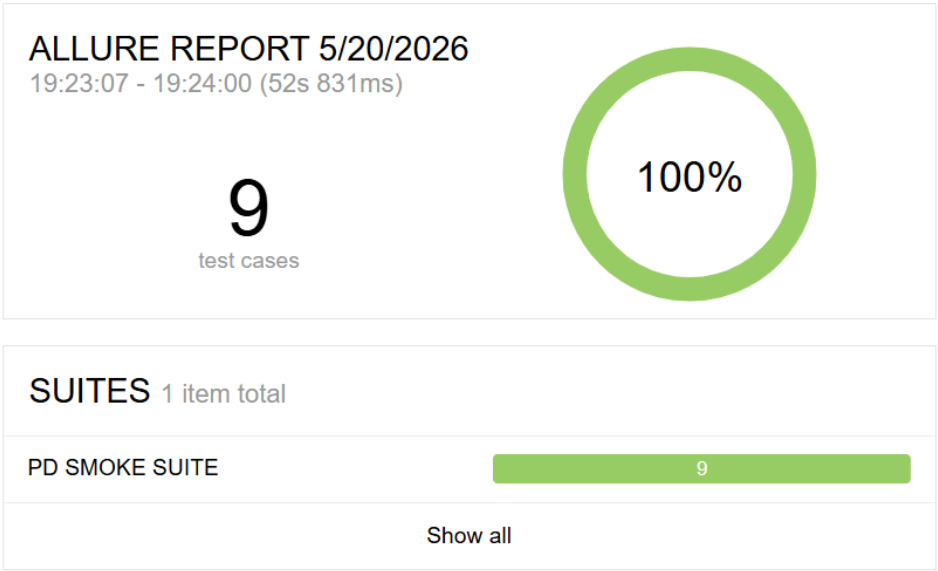


Рисунок 3.6 – Інтерфейс звіту Allure Framework

У процесі реалізації програмного додатка було також забезпечено підтримку логування виконання тестових сценаріїв. Для цього використовувалися механізми логування Java та інтеграція з Allure Attachments. Логування дозволяє зберігати

інформацію про етапи виконання тестів та спрощує процес аналізу помилок у системі автоматизації.

Результати реалізації програмного додатка свідчать про ефективність використання підходу Data-Driven Testing у системах автоматизації тестування. Використання JSON-файлів дозволило централізувати керування тестовими даними, зменшити дублювання тестового коду та підвищити масштабованість тестового фреймворку. Інтеграція з Jenkins забезпечила автоматичний запуск тестів, а використання Allure Framework дозволило реалізувати зручну систему формування звітності про результати тестування.

У результаті реалізації програмного додатка було створено масштабований фреймворк автоматизації тестування, який підтримує підхід Data-Driven Testing та забезпечує автоматичне виконання тестових сценаріїв, централізоване керування тестовими даними й формування інтерактивної звітності про результати тестування програмного забезпечення.

## 3.2 Проведення експериментального тестування та аналіз результатів

Після завершення реалізації програмного додатка інтеграції тестування на основі даних було проведено експериментальне тестування системи з метою перевірки її працездатності, оцінки ефективності використання підходу Data-Driven Testing та аналізу результатів автоматизованого виконання тестових сценаріїв. Основною метою проведення експериментального дослідження стало визначення рівня ефективності розробленого програмного рішення, перевірка стабільності роботи тестового фреймворку та оцінка можливості масштабування системи автоматизації тестування.

Експериментальне тестування проводилося у середовищі автоматизації, реалізованому з використанням мови програмування Java, фреймворків Selenium

WebDriver і TestNG, системи безперервної інтеграції Jenkins та засобу формування звітності Allure Framework. Для зберігання тестових даних використовувалися JSON-файли, які містили різні набори параметрів для виконання тестових сценаріїв. Використання Data-Driven підходу дозволило виконувати один тест із декількома наборами тестових даних без дублювання програмного коду.

На початковому етапі експериментального тестування було підготовлено тестове середовище та налаштовано автоматичний запуск тестів через Jenkins Pipeline. Після кожного оновлення програмного коду система автоматично запускала збірку Maven-проєкту, виконувала тестові сценарії та формувала звітність у Allure Framework. Такий підхід забезпечив автоматизацію процесу тестування та дозволив оцінити стабільність роботи програмного додатка в умовах безперервної інтеграції.

У процесі тестування було реалізовано набір тестових сценаріїв для перевірки функціональності вебзастосунку авторизації користувача. Тестові сценарії виконувалися з різними наборами параметрів, що зчитувалися із JSON-файлів. Для перевірки працездатності системи використовувалися як коректні, так і некоректні вхідні дані, що дозволило оцінити стабільність роботи тестового фреймворку та механізму обробки тестових даних. Приклад структури тестових даних, використаних під час експериментального тестування, наведено на рисунку 3.7.


Рисунок 3.7 – JSON-файл із тестовими даними

Під час проведення експериментального тестування особливу увагу приділено перевірці механізму параметризації тестів. Для цього використовувався DataProvider TestNG, який автоматично передавав тестові дані у тестові методи. Використання такого підходу дозволило значно скоротити кількість тестових сценаріїв та забезпечило повторне використання програмного коду.

Приклад виконання параметризованого тесту наведено на рисунку 3.8.

```
@Test(dataProvider = "userData")
public void loginTest(UserData userData) {

    loginPage.enterUsername(userData.getUsername());
    loginPage.enterPassword(userData.getPassword());
    loginPage.clickLoginButton();

    Assert.assertEquals(
        homePage.getStatus(),
        userData.getExpectedResult()
    );
}
```

Рисунок 3.8 – Виконання параметризованого тестового сценарію

У результаті проведення експериментального тестування було встановлено, що використання підходу Data-Driven Testing дозволяє значно зменшити дублювання тестового коду та спростити підтримку тестового фреймворку. Один тестовий сценарій використовувався для виконання перевірок із різними наборами параметрів, що забезпечило масштабованість системи автоматизації та підвищило ефективність процесу тестування.

Для оцінки ефективності реалізованого програмного рішення було проведено порівняння традиційного підходу до автоматизації тестування та підходу Data-Driven Testing. У процесі дослідження аналізувалися кількість тестових сценаріїв, обсяг програмного коду, час виконання тестів та складність підтримки тестового фреймворку. Результати порівняльного аналізу наведено у таблиці 3.1.

Таблиця 3.1 – Порівняння традиційного тестування та Data-Driven Testing

| Характеристика | Традиційний підхід | Data-Driven Testing |
|---|---|---|
| Кількість тестових сценаріїв | Висока | Низька |
| Дублювання коду | Значне | Мінімальне |
| Масштабованість | Обмежена | Висока |
| Підтримка тестів | Складна | Спрощена |
| Повторне використання тестів | Часткове | Повне |
| Робота з тестовими даними | Вбудована у код | Зовнішні JSON-файли |

Під час експериментального тестування також було проаналізовано результати виконання тестових сценаріїв у системі Allure Framework. Використання Allure дозволило отримати детальну статистику проходження тестів, тривалість виконання тестових сценаріїв та інформацію про помилки у разі невдалого проходження тестів. Приклад результатів тестування в Allure Framework наведено на рисунку 3.9.

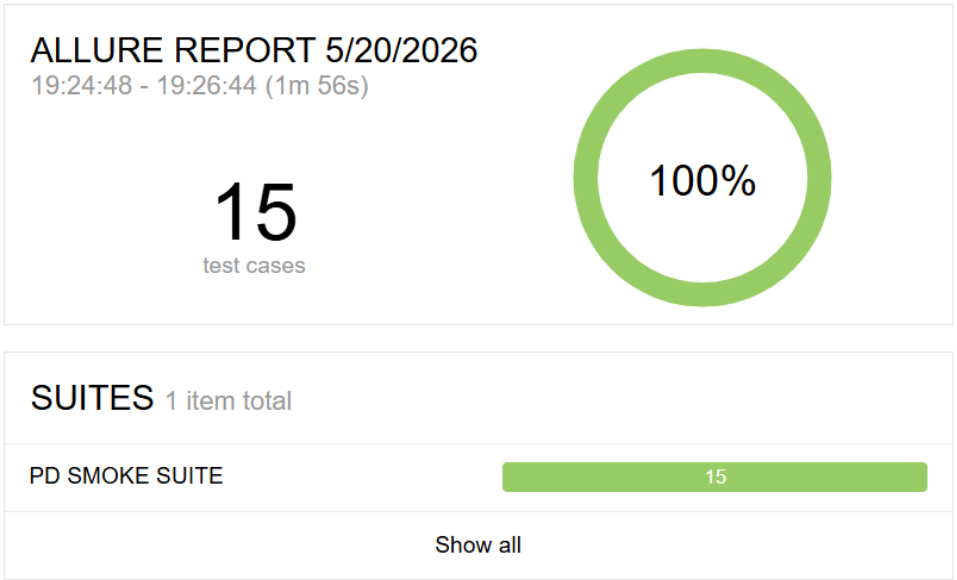


Рисунок 3.9 – Результати виконання тестів у Allure Report

У процесі аналізу результатів тестування було встановлено, що використання Jenkins забезпечує автоматизацію запуску тестів та дозволяє інтегрувати процес тестування у середовище безперервної інтеграції. Автоматичний запуск тестових сценаріїв після оновлення програмного коду дозволив скоротити час перевірки функціональності системи та підвищити стабільність процесу тестування.

Додатково було проведено аналіз ефективності роботи механізму зчитування тестових даних із JSON-файлів. У результаті тестування встановлено, що використання JSON забезпечує швидку обробку тестових параметрів та дозволяє централізовано керувати тестовими даними без необхідності зміни програмного коду тестових сценаріїв. Це суттєво спрощує підтримку системи автоматизації та дозволяє масштабувати тестовий фреймворк у разі збільшення кількості тестових сценаріїв.

У процесі проведення експериментального тестування також було перевірено стабільність роботи системи автоматизації при багаторазовому запуску тестових сценаріїв. Повторне виконання тестів із різними наборами параметрів підтвердило коректність роботи механізму параметризації та стабільність функціонування програмного додатка.

Отримані результати дослідження свідчать про ефективність використання підходу Data-Driven Testing у системах автоматизації тестування. Використання зовнішніх JSON-файлів для зберігання тестових даних дозволило забезпечити централізоване керування тестовими параметрами, зменшити дублювання тестового коду та підвищити рівень масштабованості тестового фреймворку. Інтеграція з Jenkins та Allure Framework забезпечила автоматизацію процесу запуску тестів і формування детальної звітності про результати тестування.

Проведене експериментальне тестування підтвердило працездатність розробленого програмного додатка інтеграції тестування на основі даних та доцільність використання підходу Data-Driven Testing у фреймворках автоматизації тестування програмного забезпечення.

## 3.3 Оцінка ефективності використання розробленого програмного додатка

Оцінка ефективності використання розробленого програмного додатка інтеграції тестування на основі даних є важливим етапом дослідження, який дозволяє визначити рівень продуктивності створеного тестового фреймворку, оцінити доцільність використання підходу Data-Driven Testing та встановити переваги автоматизації тестування у процесі забезпечення якості програмного забезпечення. Проведення оцінки ефективності дозволяє проаналізувати результати реалізації програмного рішення та визначити вплив використаних технологій на оптимізацію процесу автоматизованого тестування.

У процесі дослідження ефективність програмного додатка оцінювалася за такими критеріями, як швидкість виконання тестових сценаріїв, масштабованість системи автоматизації, рівень повторного використання тестового коду, зручність роботи з тестовими даними, автоматизація запуску тестів та якість формування звітності про результати тестування. Аналіз проводився шляхом порівняння традиційного підходу до автоматизації тестування та реалізованого програмного рішення на основі підходу Data-Driven Testing.

Одним із головних показників ефективності є зменшення дублювання тестового коду. У традиційних системах автоматизації для кожного набору тестових даних створюється окремий тестовий сценарій, що призводить до збільшення обсягу програмного коду та ускладнення підтримки тестового фреймворку. Використання Data-Driven Testing дозволило реалізувати механізм параметризованого тестування, у якому один тестовий сценарій може виконуватись із різними наборами параметрів, що зберігаються у JSON-файлах. Це забезпечило скорочення кількості тестових сценаріїв та підвищило рівень повторного використання програмного коду.

Результати оцінки рівня повторного використання тестових сценаріїв наведено на рисунку 3.10.


Рисунок 3.10 – Порівняння повторного використання тестових сценаріїв

Важливим критерієм ефективності стала оцінка масштабованості системи автоматизації. У процесі дослідження встановлено, що використання JSON-файлів для зберігання тестових даних дозволяє легко розширювати набір тестових параметрів без необхідності модифікації програмного коду тестових сценаріїв. Для додавання нових тестових випадків достатньо оновити JSON-файл або додати новий набір параметрів, що значно спрощує підтримку системи автоматизації.

У результаті аналізу було встановлено, що використання Data-Driven Testing дозволяє скоротити час створення нових тестових сценаріїв та зменшити складність підтримки тестового фреймворку. Крім того, централізоване керування тестовими даними забезпечує зручність роботи з параметрами тестування та дозволяє швидко оновлювати тестові набори без внесення змін у програмний код.

Для оцінки ефективності автоматизації запуску тестів було проаналізовано роботу системи Jenkins. Використання Jenkins Pipeline дозволило автоматизувати процес виконання тестових сценаріїв після внесення змін до програмного коду. Такий підхід забезпечує інтеграцію тестування у процес безперервної інтеграції та дозволяє швидко виявляти помилки у програмному забезпеченні.

Схему оцінки автоматизації тестового процесу наведено на рисунку 3.11.

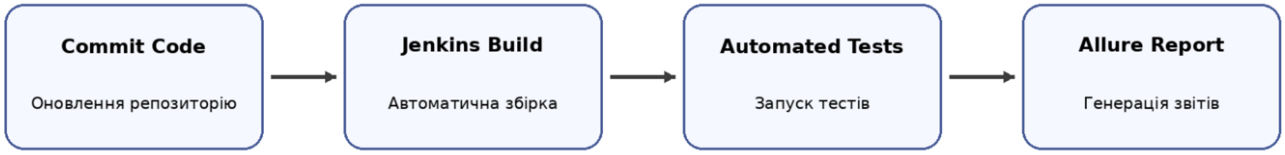


Рисунок 3.11 – Автоматизація процесу тестування

У процесі дослідження також було проведено аналіз часу виконання тестових сценаріїв. Використання автоматизованого тестування дозволило значно скоротити тривалість перевірки функціональності програмного забезпечення порівняно з ручним тестуванням. Крім того, застосування параметризованих тестів забезпечило можливість виконання великої кількості тестових перевірок без необхідності створення окремих тестових сценаріїв. Результати аналізу ефективності виконання тестів наведено у таблиці 3.2.

Таблиця 3.2 – Порівняння ручного та автоматизованого тестування

| Характеристика | Ручне тестування | Автоматизоване тестування |
|---|---|---|
| Час виконання тестів | Високий | Низький |
| Повторне використання тестів | Обмежене | Повне |
| Імовірність помилки | Висока | Мінімальна |
| Масштабованість | Низька | Висока |
| Автоматичний запуск | Відсутній | Реалізований |
| Формування звітності | Ручне | Автоматичне |

Суттєву роль у підвищенні ефективності програмного додатка відіграє система формування звітності Allure Framework. Використання Allure дозволило автоматично формувати інтерактивні звіти про результати тестування, що містять інформацію про проходження тестових сценаріїв, статистику виконання тестів, виявлені помилки та час виконання тестових перевірок. Автоматизація процесу

формування звітності значно спростила аналіз результатів тестування та підвищила зручність моніторингу стану програмного продукту.

У процесі оцінки ефективності було встановлено, що використання Java, Selenium WebDriver, TestNG, Jenkins та Allure Framework забезпечує створення масштабованої системи автоматизації тестування, яка підтримує підхід Data-Driven Testing та дозволяє централізовано керувати тестовими даними. Інтеграція всіх компонентів системи забезпечила стабільність роботи тестового фреймворку та дозволила реалізувати автоматичний запуск тестів у середовищі безперервної інтеграції.

Додатково було проаналізовано ефективність використання UML-діаграм у процесі проєктування архітектури програмного рішення. Використання UML дозволило формалізувати структуру системи, покращити розуміння взаємодії між компонентами програмного додатка та спростити документування архітектури тестового фреймворку. Це сприяло підвищенню якості проєктування системи автоматизації та забезпечило можливість подальшого масштабування програмного рішення.

У результаті проведеного дослідження встановлено, що використання механізму Data-Driven Testing дозволяє:

– зменшити дублювання тестового коду;

– підвищити повторне використання тестових сценаріїв;

– спростити підтримку тестового фреймворку;

– забезпечити масштабованість автоматизованого тестування;

– пришвидшити процес створення нових тестів;

– автоматизувати процес запуску тестових сценаріїв;

– покращити процес аналізу результатів тестування.

Запропонована архітектура програмного додатка забезпечує можливість централізованого керування тестовими даними та спрощує інтеграцію нових компонентів у систему автоматизації тестування. Використання JSON-файлів дозволяє ефективно працювати з параметризованими тестами, а інтеграція з Jenkins

та Allure Framework забезпечує автоматичний запуск тестів та формування звітності про результати тестування.

Результати оцінки ефективності підтверджують доцільність використання розробленого програмного додатка інтеграції тестування на основі даних у фреймворках автоматизації. Реалізоване програмне рішення забезпечує підвищення ефективності процесу тестування, оптимізацію роботи з тестовими даними та автоматизацію виконання тестових сценаріїв у процесі забезпечення якості програмного забезпечення.

# ВИСНОВКИ

У результаті виконання випускної кваліфікаційної роботи було розроблено програмний додаток інтеграції тестування на основі даних у фреймворк автоматизації, який забезпечує централізоване керування тестовими даними, автоматизацію виконання тестових сценаріїв та формування звітності про результати тестування.

Відповідно до поставлених завдань було отримано такі результати.

1. Проаналізовано особливості автоматизованого тестування програмного забезпечення та основні процеси його реалізації. Проведене дослідження дозволило визначити переваги автоматизації порівняно з ручним тестуванням, зокрема скорочення часу виконання тестів, зменшення впливу людського фактора та підвищення стабільності результатів перевірки програмного забезпечення.

2. Досліджено сучасні фреймворки автоматизації тестування та виконано їх порівняльний аналіз. У результаті аналізу було встановлено доцільність використання технологічного стеку Java, Selenium WebDriver, TestNG, Jenkins та Allure Framework, що забезпечило можливість створення масштабованої системи автоматизації тестування.

3. Обґрунтовано використання підходу Data-Driven Testing у системах автоматизації. Встановлено, що відокремлення тестових даних від тестових сценаріїв дозволяє мінімізувати дублювання коду, підвищити рівень повторного використання тестів та спростити підтримку тестового фреймворку в процесі його розвитку.

4. Спроєктовано архітектуру програмного додатка інтеграції тестування на основі даних. Розроблена модульна архітектура забезпечила розділення відповідальності між компонентами системи, що дозволило підвищити гнучкість програмного рішення та створити передумови для його подальшого масштабування.

5. Реалізовано програмний додаток із використанням Java та JSON-файлів для зберігання тестових даних. У результаті впровадження механізму параметризованого тестування було забезпечено можливість багаторазового використання тестових сценаріїв для різних наборів даних без зміни програмного коду тестів.

6. Інтегровано систему автоматизованого запуску тестів із Jenkins та механізм формування звітності за допомогою Allure Framework. Це дозволило автоматизувати процес виконання тестів після внесення змін до програмного коду та забезпечити отримання детальної інформації про результати тестування у вигляді інтерактивних звітів.

7. Проведено експериментальне тестування розробленого програмного додатка та виконано оцінку його ефективності. Отримані результати підтвердили, що використання підходу Data-Driven Testing забезпечує зменшення обсягу тестового коду, спрощення роботи з тестовими даними, підвищення масштабованості тестового фреймворку та скорочення часу на створення і підтримку автоматизованих тестів.

8. Обґрунтовано техніко-економічну доцільність розробки програмного додатка інтеграції тестування на основі даних у фреймворк автоматизації (додаток Б), що підтвердило економічну ефективність проекту з терміном окупності у 1,4 роки.

Практичне значення роботи полягає у створенні програмного додатка, який може бути використаний під час розробки та супроводу систем автоматизованого тестування для централізованого керування тестовими даними, автоматичного виконання тестових сценаріїв та підвищення ефективності процесу забезпечення якості програмного забезпечення. Отримані результати можуть бути використані для подальшого розвитку тестових фреймворків та вдосконалення процесів автоматизації тестування в програмних проєктах.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Панасюк І.С., Лаштун М.М. Проєктування механізму обробки тестових даних та інтеграції у фреймворк автоматизації \\ Збірник тез доповідей І Міжнародної науково-практичної конференції студентів і молодих вчених «ICSPM 2026: Інтелектуальні обчислювальні системи та управління проєктами». – Тернопіль: ЗУНУ, 2026. С. 38-40.

2. Методичні вказівки до оформлення курсових проектів, звітів про проходження практики, випускних кваліфікаційних робіт для студентів спеціальності «Комп'ютерна інженерія» / І.В. Гураль, Л.О. Дубчак / Під ред. О.М. Березького. Тернопіль: ТНЕУ, 2019. 33 с.

3. Методичні вказівки до виконання практичних робіт з дисципліни «Техніко-економічне обґрунтування розробки комп'ютерних систем»/ Н.Я. Савка, І.Р. Паздрій / Під ред. О.М. Березького. Тернопіль: ТНЕУ, 2019. 40 с.

4. Положення про організацію освітнього процесу Західноукраїнському національному університеті. Тернопіль, 2020.

5. Загальні методичні рекомендації з підготовки, оформлення, захисту та оцінювання кваліфікаційних робіт здобувачів вищої освіти першого (бакалаврського) і другого (магістерського) рівнів. Тернопіль: ЗУНУ, 2024. 83 с.

6. Методичні вказівки до випускних кваліфікаційних робіт освітнього рівня «Бакалавр» спеціальності «Комп'ютерна інженерія»/ О.М. Березький, Г.М. Мельник, Л.О. Дубчак, Ю.М. Батько / Під ред. О.М. Березького. Тернопіль: ЗУНУ, 2023. 52 с.

7. Куликовський Р. Є. Основи тестування програмного забезпечення : навч. посіб. / Р. Є. Куликовський. – Львів : Видавництво Львівської політехніки, 2021. – 256 с.

8. Бучук Ю. В. Автоматизація тестування програмного забезпечення : монографія / Ю. В. Бучук. – Київ : КПІ ім. Ігоря Сікорського, 2020. – 312 с.

9. Мельник І. О. Методи та засоби автоматизованого тестування програмних систем / І. О. Мельник // Вісник Національного університету «Львівська політехніка». Серія «Комп’ютерні науки та інформаційні технології». – 2022. – № 4. – С. 45–52.

10. Савченко В. А. Забезпечення якості програмного забезпечення в умовах автоматизованого тестування / В. А. Савченко // Комп’ютерно-інтегровані технології: освіта, наука, виробництво. – 2021. – № 42. – С. 118–124.

11. Кравченко О. П. Фреймворки автоматизації тестування вебзастосунків / О. П. Кравченко // Інформаційні технології та комп’ютерна інженерія. – 2023. – № 2. – С. 77–83.

12. Шевченко Д. М. Автоматизоване тестування як складова процесу розробки програмного забезпечення / Д. М. Шевченко // Системні дослідження та інформаційні технології. – 2020. – № 3. – С. 98–105.

13. Гнатюк С. О. Інтеграція процесів безперервного тестування у DevOps-середовище / С. О. Гнатюк // Наукові праці Чорноморського державного університету ім. Петра Могили. Серія «Комп’ютерні технології». – 2021. – Т. 334, № 322. – С. 56–63.

14. Петренко І. В. Особливості використання автоматизованих систем тестування в ІТ-проєктах / І. В. Петренко // Вісник Хмельницького національного університету. Технічні науки. – 2022. – № 6. – С. 201–207.

15. Романюк Т. В. Аналіз ефективності автоматизованого тестування програмного забезпечення / Т. В. Романюк // Інформаційні системи та мережі. – 2021. – № 9. – С. 88–95.

16. Козак Л. І. Регресійне тестування програмного забезпечення та його автоматизація / Л. І. Козак // Вісник Черкаського державного технологічного університету. – 2020. – № 1. – С. 134–140.

17. Марченко В. П. Організація процесу тестування складних програмних систем / В. П. Марченко // Наукові записки Тернопільського національного педагогічного університету. Серія: Інформатика. – 2022. – № 1. – С. 61–68.

18. Бондаренко С. І. Використання Selenium та TestNG для автоматизації тестування вебзастосунків / С. І. Бондаренко // Комп’ютерні науки та кібербезпека. – 2023. – № 3. – С. 73–80.

19. Литвиненко М. О. Підхід Data-Driven Testing у системах автоматизації тестування / М. О. Литвиненко // Інженерія програмного забезпечення. – 2021. – № 2. – С. 91–97.

20. Ковальчук А. М. Автоматичне формування звітності результатів тестування програмного забезпечення / А. М. Ковальчук // Управляючі системи та машини. – 2022. – № 5. – С. 49–55.

21. Остапенко Р. В. Безперервна інтеграція як складова сучасної розробки програмного забезпечення / Р. В. Остапенко // Комп’ютерно-інтегровані технології та системи. – 2021. – № 4. – С. 112–118.

22. Тимошенко Н. С. Архітектурні підходи до побудови фреймворків автоматизованого тестування / Н. С. Тимошенко // Сучасні інформаційні технології у сфері безпеки та оборони. – 2023. – № 1. – С. 85–92.

23. Бучук Ю. В. Автоматизація тестування програмного забезпечення : монографія / Ю. В. Бучук. – Київ : КПІ ім. Ігоря Сікорського, 2020. – 312 с.

24. Fewster M. Software Test Automation / M. Fewster, D. Graham. – Boston : Addison-Wesley, 1999. – 464 p.

25. Кравченко О. П. Фреймворки автоматизації тестування вебзастосунків / О. П. Кравченко // Інформаційні технології та комп’ютерна інженерія. – 2023. – № 2. – С. 77–83.

26. Holmes A. Selenium Testing Tools Cookbook / A. Holmes. – Birmingham : Packt Publishing, 2015. – 454 p.

27. Microsoft Corporation. Playwright Documentation [Електронний ресурс]. – Режим доступу: https://playwright.dev/

28. Jain A. REST Assured Automation Framework / A. Jain. – New Delhi : BPB Publications, 2021. – 356 p.

29. Бондаренко С. І. Використання Selenium та TestNG для автоматизації тестування вебзастосунків / С. І. Бондаренко // Комп’ютерні науки та кібербезпека. – 2023. – № 3. – С. 73–80.

30. Литвиненко М. О. Підхід Data-Driven Testing у системах автоматизації тестування / М. О. Литвиненко // Інженерія програмного забезпечення. – 2021. – № 2. – С. 91–97.

31. Fewster M. Software Test Automation / M. Fewster, D. Graham. – Boston : Addison-Wesley, 1999. – 464 p.

32. Куликовський Р. Є. Основи тестування програмного забезпечення : навч. посіб. / Р. Є. Куликовський. – Львів : Видавництво Львівської політехніки, 2021. – 256 с.

33. Мельник І. О. Методи та засоби автоматизованого тестування програмних систем / І. О. Мельник // Вісник Національного університету «Львівська політехніка». Серія «Комп’ютерні науки та інформаційні технології». – 2022. – № 4. – С. 45–52.

34. Holmes A. Selenium Testing Tools Cookbook / A. Holmes. – Birmingham : Packt Publishing, 2015. – 454 p.

35. Meszaros G. xUnit Test Patterns: Refactoring Test Code / G. Meszaros. – Boston : Addison-Wesley, 2007. – 880 p.