Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра автоматизації, комп’ютерних наук і технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти – бакалавр
за освітньо-професійною програмою
«Комп’ютерні науки»
зі спеціальності
122 – Комп’ютерні науки

тема роботи:

«Алгоритмічне забезпечення інтегрованого середовища автоматизованого тестування програмних продуктів»

Виконав студент гр. КН-21 _____ Новохатько О. О.

Керівник _____ Жосан А. А.

Нормоконтроль _____ Маринич І. А.

Завідувач кафедри _____ Рубан С. А.

Кривий Ріг – 2025

КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет: *інформаційних технологій*

Кафедра: *автоматизації, комп'ютерних наук і технологій*

Ступінь вищої освіти: *Бакалавр*

Спеціальність: *122 – Комп'ютерні науки*

ЗАТВЕРДЖУЮ

Зав. кафедрою: *к.т.н. Рубан С.А.*

_____

« *29* » *січня* 2025 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра

студентові групи КН-21 *Новохатько Олегу Олександровичу*

1. Тема кваліфікаційної роботи: *«Алгоритмічне забезпечення інтегрованого середовища автоматизованого тестування програмних продуктів»*

затверджено наказом по університету № *254с* від *13.05.2025 р.*

2. Термін здачі кваліфікаційної роботи: *06.06.2025 р.*

3. Склад кваліфікаційної роботи: *Пояснювальна записка обсягом 62с., додатки, презентація у Microsoft PowerPoint (13 слайдів) в електронному та друкованому вигляді*

4. Консультанти кваліфікаційної роботи:

Розділ 1-2 доц. Жосан А. А.

Нормоконтроль доц. Маринич І. А.

5. Календарний план:

№	Етапи роботи	Термін виконання
1	*Вступ*	*01.04.25*
2	*Розділ 1*	*05.04.25*
3	*Розділ 2*	*01.05.25*
4	*Висновки*	*25.05.25*
5	*Оформлення кваліфікаційної роботи*	*28.05.25*
6	*Підготовка презентації та графічного матеріалу*	*20.05.25*
7	*Підготовка доповіді до захисту*	*05.06.25*

6. Дата видачі завдання: *28.01.2025р.*

Керівник _____ / Жосан А. А./

7. Запевнення: *Я, Новохатько Олег Олександрович, запевняю, що ця кваліфікаційна робота виконана самостійно, не містить академічного плагіату, фабрикації, фальсифікації. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.*

Із чинним Положенням про академічну доброчесність Криворізького національного університету ознайомлений.

Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі умисних порушень робота не допускається до захисту або оцінюється незадовільно.

Студент _____ / Новохатько О. О./

АНОТАЦІЯ

Новохатько О. О. Алгоритмічне забезпечення інтегрованого середовища автоматизованого тестування програмних продуктів : кваліфікаційна робота бакалавра : 122 – Комп’ютерні науки. Кривий Ріг. Криворізький національний університет, 2025. 62 с.

Проектування якісного програмного забезпечення передбачає використання великої кількості тестів, розробка та виконання яких є повільним і трудомістким процесом. Розробка моделей і алгоритмів для інтегрованого середовища автоматизованого тестування програмного забезпечення є актуальною задачею.

Метою роботи є вивчення та розробка моделей та алгоритмів інтегрованого середовища автоматизованого тестування програмного забезпечення, що забезпечує підвищення ефективності процесу тестування програмного забезпечення.

У вступі обґрунтовується актуальність теми роботи, сформульована мета та завдання для її досягнення.

У першому розділі проаналізовано стан досліджень у галузі побудови інтегрованих автоматизованих середовищ тестування програмного забезпечення та методичні підходи до його побудови.

Другий розділ присвячений розробці моделей і алгоритмів інтегрованого середовища автоматизованого тестування програмного забезпечення, а також апробації запропонованих конструктивних рішень.

Практична цінність одержаних результатів полягає в можливості практичного застосування розробленого інтегрованого середовища для підвищення ефективності тестування програмного забезпечення.

Ключові слова:

АВТОТЕСТ, АВТОМАТИЗАЦІЯ ТЕСТУВАННЯ, АЛГОРИТМ, ІНТЕГРОВАНЕ СЕРЕДОВИЩЕ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ФРЕЙМВОРК, JUNIT, SELENIUM, CUCUMBER

ЗМІСТ

ВСТУП

Проектування якісного програмного забезпечення передбачає використання великої кількості тестів, розробка та виконання яких є повільним і трудомістким процесом.

Як показує практика, висока ефективність процесу тестування програмного забезпечення досягається за допомогою його автоматизації, що, крім підвищення продуктивності, дозволяє знизити негативний вплив людського фактору на результати тестування. Автоматизоване тестування програмного забезпечення пропонує використання інструментів, за допомогою яких тестувальники IT-компаній не будуть витрачати час та інші ресурси на монотонні завдання, оскільки автоматизовані тести можуть виконуватися безперервно або за розкладом через певні проміжки часу.

Для вирішення цієї проблеми використовуються інтегровані середовища автоматизованого тестування програмного забезпечення.

Розробка моделей і алгоритмів для такого середовища є актуальною задачею і становить і практичний інтерес.

Об'єктом даної роботи є інтегроване середовище автоматизованого тестування програмного забезпечення, а предметом роботи є моделі та алгоритми інтегрованого середовища автоматизованого тестування програмного забезпечення.

Метою роботи є вивчення та розробка моделей та алгоритмів інтегрованого середовища автоматизованого тестування програмного забезпечення, що забезпечує підвищення ефективності процесу тестування програмного забезпечення.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– виконати огляд та аналіз існуючих рішень для інтегрованих автоматизованих середовищ тестування програмного забезпечення;

– проаналізувати методологічні підходи до побудови інтегрованих автоматизованих середовищ тестування програмного забезпечення;

– розробляти моделі та алгоритми для інтегрованого середовища автоматизованого тестування програмного забезпечення;

– виконати тестування та оцінювати ефективність інтегрованого середовища автоматизованого тестування програмного забезпечення, реалізованого на основі розроблених моделей та алгоритмів.

Основна цінність роботи полягає в розробці моделей і алгоритмів інтегрованого автоматизованого тестового середовища, що забезпечує автоматизацію тестування програмного забезпечення.

А практична цінність одержаних результатів полягає в можливості практичного застосування розробленого інтегрованого середовища автоматизованого тестування для підвищення ефективності тестування програмного забезпечення в ІТ-компаніях.

РОЗДІЛ 1
АНАЛІЗ СУЧАСНОГО СТАНУ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Аналіз методів автоматизованого тестування

Автоматизоване тестування – це метод тестування (програмне забезпечення), який виконується за допомогою спеціальних програмних засобів для виконання набору тест-кейсів.

Проблеми автоматизації тестування програмного забезпечення розглядаються в роботах Е. Дастіна, М. Кона, М. Фаулера, фахівців вендорів програмного забезпечення та ін. Загальноприйнятим вважається наступне визначення автоматизованого тестування програмного забезпечення - це процес верифікації програмного забезпечення, в якому основні функції та етапи тестування, такі як запуск, ініціалізація, виконання, аналіз та повернення результату, виконуються автоматично за допомогою інструментів автоматизованого тестування [1].

Автоматизація тестування надає наступні переваги:

- економія коштів у порівнянні з ручним тестуванням;
- покращення якості за рахунок виключення людського фактору, повторного використання автотестів на різних етапах розробки, регулярної перевірки функціональності в процесі розробки та забезпечення того, щоб QA-спеціалісти могли приділяти більше уваги перевіркам, які не охоплені автотестами;
- оптимізація обсягу тестування – можливість протестувати більше функціоналу за той же час, а також застосувати більш креативні методи тестування;
- скорочення часу тестування – регулярні регресійні тести можуть використовуватися для швидкого виявлення старих дефектів, а скорочення часу завдяки автоматизації дозволяє швидше вивести ваше програмне

забезпечення на ринок.

Автоматизоване тестування було запропоновано М. Коном, який представив процес автоматизації системи і перевірки програмного забезпечення у вигляді піраміди. Структура піраміди випробувань показана на рисунку 1.1 [2].

Рисунок 1.1 – Піраміда автоматизованого тестування програмного забезпечення

Згідно з цим підходом, розроблена ефективна стратегія автоматизації тестування передбачає автоматизацію тестування на трьох різних рівнях.

Рівень тестування - це група тестових дій, заснованих на загальних характеристиках, пов'язаних з життєвим циклом розробки програмного забезпечення [3].

В основі піраміди лежить модульне тестування.

Модульне тестування має бути основою надійної стратегії автоматизації тестування і тому становить найбільшу частину піраміди. Автоматизовані модульні тести ефективні, оскільки надають програмісту конкретні дані. Крім того, оскільки модульні тести зазвичай пишуться на тій же мові, що і система, програмістам часто зручніше їх писати.

Інтеграційне тестування – це спосіб тестування програмного забезпечення підтримки шляхом групування програмних компонентів. Автоматизоване інтеграційне тестування проводиться для оцінки відповідності додатку або його компонентів заданим функціональним вимогам.

Автоматизоване наскрізне тестування (end-to-end або E2E тестування) – це тестування інтерфейсу користувача, який знаходиться на вершині піраміди. Слід зазначити, що наскрізне тестування коштує дорого і має бути зведене до мінімуму.

Елементи піраміди Кона відрізняються один від одного обсягом тестування, так як містять різну кількість тест-кейсів, необхідних для виконання того чи іншого виду тестування.

Представлена піраміда характерна для тестування програмного забезпечення. Але існує безліч його модифікацій - все залежить від типу тестуємого додатку» [4].

Веб-додатки динамічні й інтерактивні у порівнянні з традиційним додатками. Отже, традиційних методів та інструментів тестування недостатньо для тестування веб-додатків. Тому ефективність роботи команди тестування багато в чому залежить від того, які завдання було вирішено автоматизувати і як проводилася ця автоматизація.

1.2 Огляд та аналіз джерел за темою роботи

Автоматизоване тестове середовище – це набір засобів автоматизації тестування, інтегрованих у середовище розробки програмного забезпечення. Girgis M.R. et al. пропонують підхід до веб-тестування, при якому гіперпосилання тестованого веб-сайту автоматично слідують один за одним, щоб отримати всі HTML-тексти його сторінок, починаючи з домашньої сторінки. Аналізується HTML-текст кожної виявленої сторінки, щоб витягнути необхідну інформацію про неї. Зібрана інформація потім

використовується в перевірка помилок [5]. Архітектура автоматизованої системи тестування, що реалізує описаний підхід, показана на рисунку 1.2.

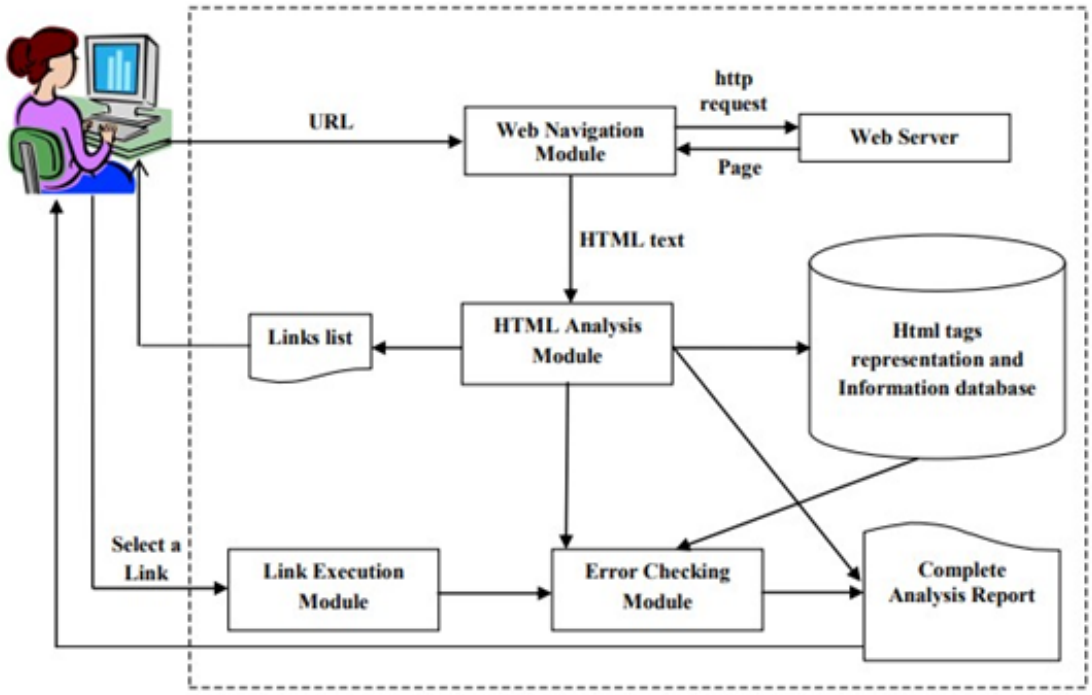

Рисунок 1.2 – Архітектура автоматизованої системи тестування веб-додатків

На думку авторів рішення, запропонований підхід гарантує відповідність двом критеріям тестування веб-додатків, а саме критерію охоплення сторінки та критерію покриття гіперпосиланнями.

У багатьох рішеннях для автоматизації тестування використовуються фреймворки.

Фреймворк автоматизованого тестування – це сукупність умов, концепцій та практик, спрямованих на повторне використання, зниження витрат на підтримку та підвищення надійності використання тестів [6].

Як зазначає М. Фаулер, піраміда випробувань також являє собою проміжний рівень тестів, які функціонують через сервісний рівень програми. Вони можуть надати багато переваг наскрізних тестів, але вони дозволяють уникнути багатьох складнощів роботи з UI-фреймворками інтерфейсу.

У веб-додатках це буде відповідати тестуванню через шар API, в той час як вершина піраміди користувацького інтерфейсу буде відповідати тестам з використанням чогось на кшталт Selenium або Sahi [6].

B. Garcıa та J.C. Duenas пропонують метод автоматизації тестування, який виконується на чотирьох рівнях: генерація тест-кейсів, збір тестових даних, виконання тест-кейсів та звітування про тестові випадки.

Цей метод базується на трьох типах вхідних даних: UML-моделях, скриптах Selenium і XML-файлах. Підхід реалізований в середовищі тестування з відкритим вихідним кодом під назвою Automatic Testing Platform [7].

M. Hanna et al. пропонують фреймворк автоматизованого тестування веб-додатків, який, на їхню думку, поліпшить процес автоматизації (рис. 1.3) [8].

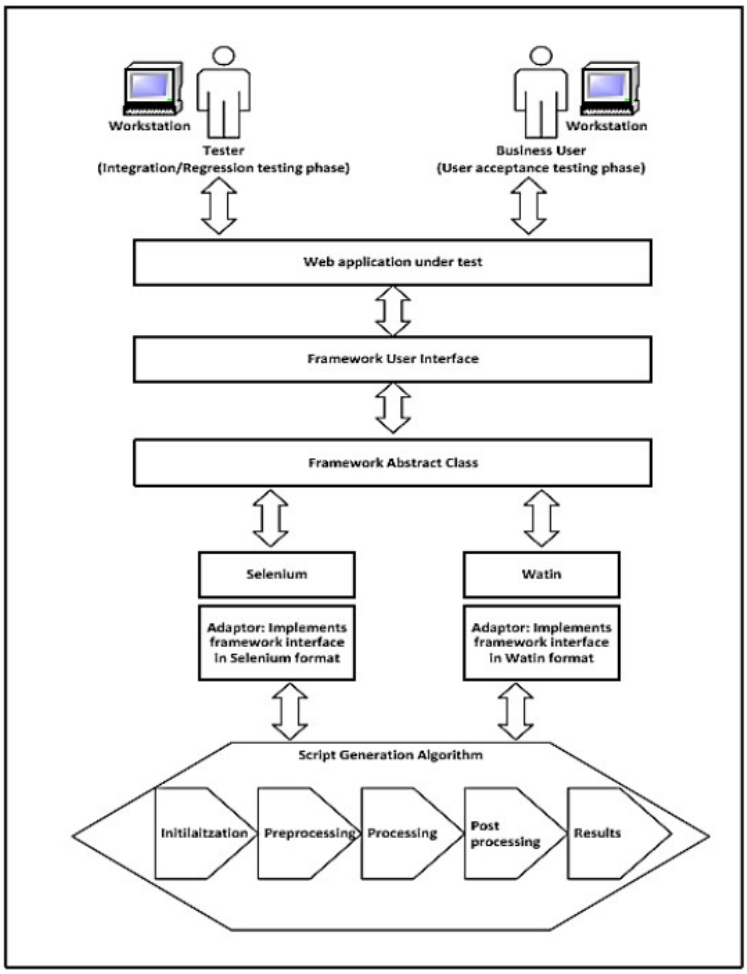

Рисунок 1.3 – Модель фреймворку тестування веб-додатків

У дослідженні M. Kalmo описаний експеримент з оцінки впливу фрейворку на ефективність тестування веб-додатків, розроблених на мові Java [9]. Суть цього експерименту полягає в тому, щоб зібрати два прототипи

додатку для автоматизації тестування - з підтримкою фреймворку і без неї. Як показав порівняльний аналіз, використання фреймворків значно підвищує ефективність тестування веб-додатків на всіх рівнях піраміди тестування.

Таким чином, найбільш ефективним підходом до автоматизації тестування веб-додатків є використання спеціалізованих фреймворків.

У [10] описана інтегроване середовище автоматизації тестування, побудована на основі технології Eclips.

Це рішення дозволяє нам забезпечити незалежність платформи та максимально спростити роботу з налаштування та обслуговування. Інтегроване середовище складається з наступних модулів (рис. 1.4):

– парсер, призначений для імпорту та експорту даних для налаштування тестового оточення з внутрішньої моделі даних;

– модель даних, що дозволяє зберігати інформацію з конфігураційного файлу в спеціалізованій моделі Eclipse;

– користувальницький інтерфейс на основі використання технології GEF (Graphical Editing Framework), що дає можливість швидкого введення та редагування даних;

– менеджер стартапів – модуль, що дозволяє легко збирати довільні ланцюжки інструментів ТАТ і допоміжних скриптів.

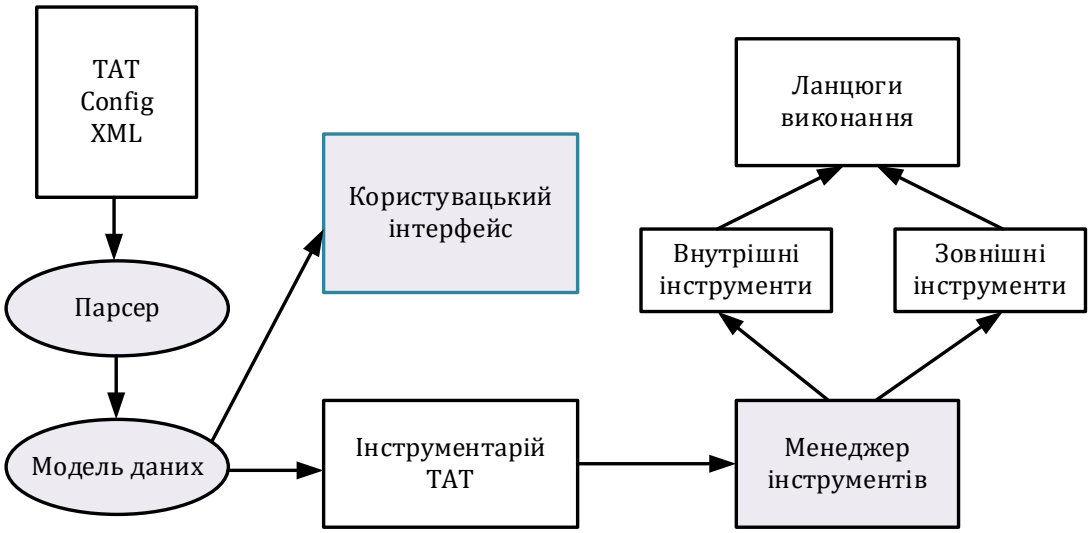

Рисунок 1.4 – Архітектура інтегрованого середовища автоматизації тестування

Разом з тим, необхідно констатувати відсутність робіт, присвячених розробці моделей та алгоритмів для інтегрованого середовища автоматизованого тестування програмного забезпечення, що підтверджує актуальність теми дослідження.

1.3 Огляд та аналіз аналогів інтегрованого середовища тестування програмного забезпечення

За своїми функціональними та архітектурними особливостями інтегроване середовище автоматизованого тестування (ICAT) програмного забезпечення належить до середовищ автоматизованого тестування [10].

Середовище автоматизованого тестування (CAT) - це комбінація апаратного забезпечення, програмного забезпечення, даних і конфігурації, необхідних для виконання тестових випадків. Для розробки вимог до ІС ми використовуємо методологію FURPS+.

FURPS – це метод перевірки пріоритетних вимог після розуміння потреб клієнта. Методологія FURPS у класифікації вимог наголошує на розумінні різних типів функціональних та нефункціональних вимог. Функціональні вимоги описують, що повинна робити ІС, тоді як нефункціональні вимоги накладають обмеження на те, як система ІС буде це робити [11]. У таблиці 1.1 представлені основні вимоги до програмного забезпечення ICAT в методології FURPS.

Розроблений набір вимог є основою для реалізації проектного рішення з розробки програмного забезпечення ICAT.

У процесі аналізу відомих CAT-рішень були виявлені наступні інструменти автоматизованого тестування веб-додатків: Sahi Pro, Selenium IDE, Watir.

Нижче розглянемо характеристики цих рішень .

Таблиця 1.1 – Вимоги до ІС тестування веб-додатків

№	Вимога	Статус	Корисність	Ризик	Стабільність
Functionality - Функціональні вимоги					
1	Забезпечення автоматизації тестів на 3-х рівнях тестування веб-додатків	схвалене	критична	середній	низька
Usability – Вимоги зручності користування					
2	Дружній, інформативний інтерфейс	схвалене	критична	середній	низька
Reliability – Вимоги надійності					
3	Допустима частота збоїв: 1 раз в 300г.	схвалене	важлива	середній	середня
4	Середній час збоїв: 1 роб. доба	схвалене	важлива	середній	середня
5	Можливість відновлення системи після збоїв: 1 роб. доба	схвалене	важлива	середній	середня
6	Режим роботи: 7/24/365	схвалене	важлива	середній	середня
Performance – Вимоги продуктивності					
7	Допустима кількість одночасно працюючих користувачів:1	запропоноване	важлива	середній	середня
8	Час реакції на виникнення аварійної ситуації: 10 с	запропоноване	важлива	середній	середня
Supporstbility – Вимоги підтримки					
9	Час усунення критичних проблем: протягом робочої доби	запропоноване	важлива	середній	середня

1.3.1 Sahi Pro

Sahi Pro – це набір готових до використання інструментів для автоматизованого тестування веб-додатків, веб-сервісів, мобільних пристроїв,

Windows Desktop, SAP GUI та Java додатків [12]. Допомагає автоматизувати функціональне тестування веб-додатків. Sahi Pro за замовчуванням підтримує автоматизацію веб-додатків і REST API. Sahi використовує проксі для вставки JavaScript у веб-сторінки (рисунок 1.5).

Sahi Pro добре підходить для кросбраузерного тестування складних додатків Web 2.0 з великою кількістю AJAX та динамічного контенту. Sahi Pro працює в будь-якому сучасному браузері, який підтримує javascript.

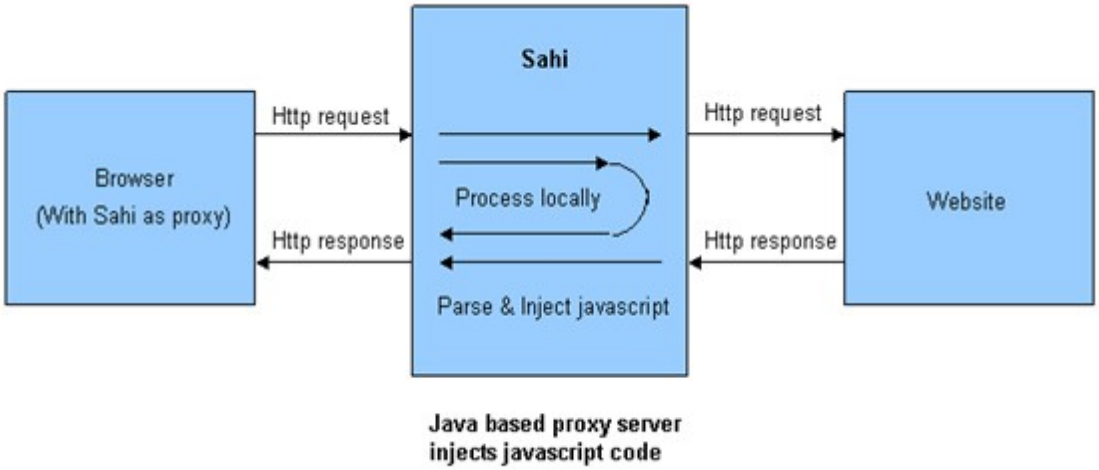

Рисунок 1.5 – Архітектура CAT Sahi Pro

1.3.2 Selenium IDE

Просте у використанні розширення для браузера Firefox, яке допомагає розробляти тестові сценарії для веб-сторінок [13]. Він фіксує певний сценарій поведінки користувачів на сайті, а потім автоматично відтворює записані дії. Selenium IDE також має понад 500 вбудованих команд, які дозволяють записати практично будь-які ваші дії на сайті і відтворити їх ще раз або перевірити якийсь елемент.

Selenium IDE можна розширити за допомогою плагінів. Вони можуть впроваджувати нові команди в IDE або інтегруватися зі стороннім сервісом. Робочий процес в Selenium IDE показаний на рисунку 1.6.

Основні переваги:

– просте, готове рішення для швидкого створення надійних

наскрізних тестів. Готовий працювати з будь-яким веб-додатком;

– Просте налагодження тестів з багатими функціями IDE, такими як установка брейкпойнтів і призупинення винятків.

– можливість паралельного запуску тестів у будь-якій комбінації браузер/ОС за допомогою командного рядка.

Рисунок 1.6 – Потік робіт в Selenium IDE

Недоліки:

– надмірна простота проведення автотестів;

– Обмежене використання (для браузерів Firefox та Chrome).

1.3.3 Watir

Watir – це інструмент тестування з відкритим вихідним кодом, розроблений з використанням Ruby, який використовується для тестування веб-додатків, розроблених на будь-якій мові програмування [14]. Архітектура Watir показана на рисунку 1.7.

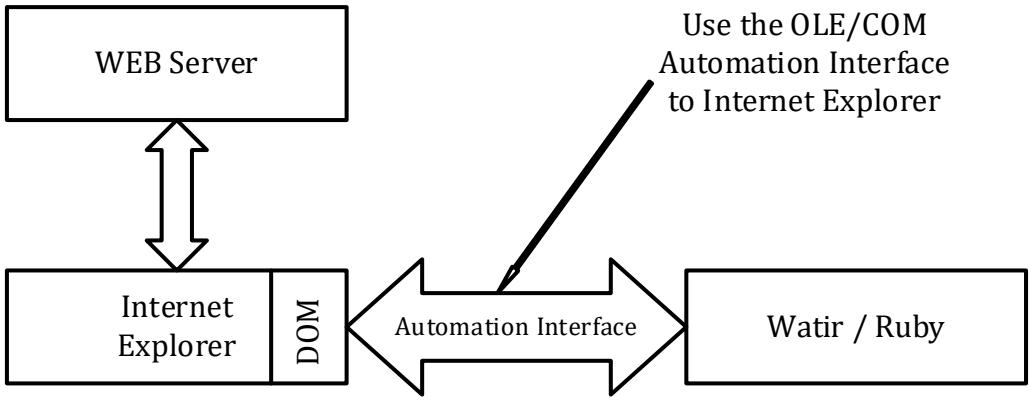

Рисунок 1.7 – Архітектура CAT Watir

Основні функції:

– тестує мовний веб-додаток;

– кросбраузерне тестування;

– перевіряє кнопки, форми, посилання та відповіді на них на веб-сторінках. Основним недоліком є те, що Watir підтримує тільки тестові середовища Ruby і не може використовуватися з іншими тестовими системами.

Для порівняльного аналізу вищевказаних CAT-рішень на відповідність вимогам, що пред'являються до програмного забезпечення ICAT, ми використовуємо таблицю 1.2.

Таблиця 1.2 – Порівняльний аналіз CAT [15]

Характеристика/бал (0-3)	Sahi Pro	Selenium IDE	Watir
Забезпечення автоматизації тестів на 3-х рівнях тестування веб-додатків	2	2	2
Застосування сучасних IT	3	3	3
Низька сукупна вартість	2	3	3
Підсумок	7	8	8

Таким чином, жодне з розглянутих готових CAT-рішень не відповідає всім встановленим вимогам, що підтверджує доцільність розробки програмного забезпечення ICAT.

1.4 Вибір методики проектування інтегрованого середовища автоматизованого тестування програмного забезпечення

Згідно з концепцією Agile дизайну, автоматизоване тестування має бути не ізольованим завданням, а безперервним процесом, який за своєю суттю вписаний у життєвий цикл програмного забезпечення.

Загалом стратегія автоматизованого тестування включає загальний підхід до тестування та звітності. Всі процеси намічаються і плануються разом з очікуваннями від циклу тестування [14].

Стратегія автоматизації тестування веб-додатків складається з

наступних етапів:

Стадія 1. Вибір фреймворку тестування для кожного рівня автоматизації тестування. Іншими словами, необхідно виділити і намітити спектр використовуваних інструментів, а також їх обсяг. При створенні CAT розробник повинен забезпечити його масштабованість. Слід зазначити, що фреймворк рідко використовується для одного проекту - замість цього він є шаблоном для стратегії автоматизації тестування, інструкцій і принципів, які тестувальники повинні вміти використовувати для будь-якого виду автоматизованого тестування.

Стадія 2. Розробка автоматизованого середовища тестування. Хороша CAT повинна, перш за все, повністю контролюватися і давати тестувальнику свободу змінювати всі необхідні дані.

Стадія 3. Налаштування CAT. Це частина, де тестувальник визначає обсяг тесту, створює список дій і призначає членів команди для нагляду за кожним кластером. Стандартний функціональний набір автоматизованого тестування зазвичай виглядає так: «Ініціювання», «Автоматизоване планування тестування», «Виконання», «Звітність», «Завершення».

Стадія 4. Впровадження CAT в процес тестування веб-додатків IT-компанії.

Незважаючи на те, що структура стратегії багато в чому залежить від структури IT-компанії, кількості залучених людей та методології проектування, прийнятої в IT-компанії, кількість етапів та дій, які необхідно виконати, залишається досить стабільною.

Розглянемо та порівняємо відомі фреймворки за рівнями автоматизації тестування веб-додатків.

1.4.1 Модульне тестування програмного забезпечення

Щоб вибрати фреймворк для модульного тестування, давайте порівняємо характеристики фреймворків Jasmine, Mocha і JUnit.

Фреймворк Jasmine — це поведінкове середовище розробки для

тестування JavaScript-коду. Jasmine не залежить від інших фреймворків JavaScript і не вимагає DOM.

Переваги:

– Простота: Jasmine відомий своїм простим синтаксисом і зручністю використання. Це чудовий вибір для невеликих наборів тестів, де простота має значення;

– Розвиток на основі поведінки (BDD): ця читабельність покращує співпрацю між членами команди;

– Комплексне моделювання та затвердження. Вбудована функція SpyOn дозволяє імітувати дзвінки, що робить її корисною для тестування асинхронного коду.

Недоліки:

– зовнішні залежності: Jasmine використовує зовнішні бібліотеки (наприклад, Chai) для охоплення всіх аспектів тестування, що означає додаткову конфігурацію та обслуговування;

– Обмежені можливості. У порівнянні з деякими іншими фреймворками, такими як Jest, Jasmine не вистачає певних вбудованих функцій;

– повільне виконання. Швидкість виконання тестів Jasmine може бути нижчою, ніж у Jest через те, що він покладається на зовнішні бібліотеки.

Підводячи підсумок, можна сказати, що Jasmine є хорошим вибором для простих наборів тестів і сценаріїв, де читабельність і принципи BDD мають вирішальне значення.

Фреймворк Mocha — це багатофункціональний тестовий фреймворк JavaScript, який працює на Node.js та в браузері, що робить асинхронне тестування простим і цікавим.

Тести Mocha виконуються послідовно, забезпечуючи гнучку та точну звітність і відображаючи невиявлені винятки в правильних тестових випадках. Розміщено на GitHub. Mocha можна використовувати з більшістю відомих

бібліотек тверджень JavaScript, включаючи Chai. Тести в Mocha мають покращену якість відстеження винятків і можуть виконуватися серіями [2].

Переваги:

– Гнучкість і налаштування: Mocha надає гнучку архітектуру, яка дозволяє розробникам адаптувати своє тестове середовище відповідно до потреб конкретного проекту.

– Зовнішнє тестування: Mocha відмінно справляється з тестуванням внутрішнього коду. Його універсальність дозволяє легко інтегруватися з іншими бібліотеками для створення всеосяжного набору тестів;

– багата екосистема: Mocha має велику екосистему плагінів і розширень.

Недоліки:

– Налаштування над головними витратами На відміну від деяких фреймворків на кшталт Jest, які мають вбудовані функції, Mocha вимагає додаткової конфігурації;

– крива навчання. Гнучкість Mocha походить від більш крутої кривої навчання;

Таким чином, вибирайте Mocha, якщо ви цінуєте кастомізацію і вам потрібне потужне тестове середовище для бекенд-проектів.

Фреймворк JUnit – це фреймворк модульного тестування для мови програмування Java. Він відіграє вирішальну роль у розробці на основі тестування та є сімейством фреймворків модульного тестування, відомих під загальною назвою xUnit. JUnit просуває ідею «спочатку тестуй, а потім коду», яка наголошує на налаштуванні тестових даних для шматка коду, який можна спочатку протестувати, а потім впровадити. [31].

Переваги:

– відкритий вихідний код: JUnit — це платформа з відкритим вихідним кодом, що робить її доступною для розробників без будь-яких витрат на ліцензування;

– Agile-тестування: JUnit дозволяє писати тест-кейси для різних сценаріїв, включаючи модульні тести, інтеграційні тести та системні тести.

– читабельність та виразність;

– автоматизоване тестування;

– GUI та CLI: JUnit пропонує як текстові інтерфейси командного рядка, так і графічні інтерфейси (AWT та Swing).

Недоліки:

– тестування обмеженої залежності. На відміну від фреймворків на кшталт TestNG, JUnit не має вбудованої підтримки тестування залежностей;

– не ідеально підходить для великих наборів тестів: JUnit може бути не найкращим вибором для великих наборів тестів;

– без групового тестування. JUnit не має вбудованої підтримки групування тестів, функції, доступної в TestNG;

– відсутність HTML-звітів. JUnit не генерує HTML-звіти для тест-кейсів, які можуть бути корисними для візуального відстеження результатів тестування.

Підводячи підсумок, можна сказати, що JUnit є надійним вибором для модульного тестування Java, особливо для невеликих проектів.

Для порівняльного аналізу характеристик засобів автоматизації модульного тестування веб-додатку використовуємо таблицю 1.3.

Таблиця 1.3 – Порівняльний аналіз фреймворків модульного тестування веб-додатків [16]

Характеристика (0-5)	JUnit	Jasmine	Mocha
Простота налаштування	5	3	2
Простота освоєння	3	5	4
Ком’юніті	4	5	3
Функціональність	5	4	3
Документація	5	5	4
Продуктивність	5	2	2
Підсумок	27	24	18

Як показав аналіз, фреймворк JUnit має найкращі характеристики, що повністю пояснює його популярність серед тестувальників. Ще однією важливою перевагою JUnit є простота інтеграції в різні інструменти розробника (наприклад, IntelliJ).

1.4.2 Інтеграційне програмного забезпечення

Щоб вибрати фреймворк для інтеграційного тестування, давайте порівняємо характеристики фреймворків JWebUnit, Cucumber і Protractor.

Фреймворк JWebUnit — це фреймворк для тестування на основі Java та одне з розширень JUnit, яке використовується для інтеграції, регресії та функціонального тестування. Завдяки цьому ви можете відразу тестувати свої веб-додатки.

Переваги:

– створення приймально-здавальних випробувань: JWebUnit спеціально розроблений для створення приймально-здавальних тестів для веб-додатків;

– інтеграція з HtmlUnit: JWebUnit використовує HtmlUnit для веб-взаємодії;

– читабельність і рефакторинг. Як і у випадку з рефакторингом, JWebUnit надає читабельний тестовий код.

Недоліки:

– зовнішні залежності: JWebUnit використовує зовнішні бібліотеки, такі як HtmlUnit та JUnit;

– крива навчання. Розробники, які працюють з JWebUnit вперше, можуть зіткнутися з труднощами в навчанні через його специфічну інтеграцію з HtmlUni;

Таким чином, JWebUnit є підходящим вибором для приймального тестування веб-додатків Java, особливо коли потрібна інтеграція з HtmlUnit.

Фреймворк Cucumber – це фреймворк, який реалізує підхід BDD/TDD.

Переваги:

– Behavior-Driven Development (BDD): це фреймворк BDD, який заохочує співпрацю між розробниками, тестувальниками та нетехнічними зацікавленими сторонами;

– читабельний і виразний синтаксис. Синтаксис Cucumber написан в зручному для читання форматі з використанням мови Gherkin;

– повторне використання: сприяє повторному використанню, дозволяючи використовувати покрокові визначення в різних сценаріях.

– багатомовна інтеграція: підтримує кілька мов програмування (таких як Java, Ruby та JavaScript), що робить його універсальним для різних технологічних стеків;

– звіти про тестування та візуалізація: генерує детальні звіти про тестування, включаючи інформацію про пройдені та невдалі сценарії.

Недоліки:

– налаштування Cucumber включає налаштування файлів функцій, визначень кроків;

– повільне виконання. Скрипти Cucumber можуть виконуватися повільніше, ніж традиційні модульні тести;

– проблеми з обслуговуванням. У міру зростання проекту підтримка скриптів Cucumber стає складнішою;

– обмежене паралельне виконання: за своєю суттю не підтримує паралельне виконання скриптів.

Підводячи підсумок, можна сказати, що Cucumber цінний з точки зору співпраці, читабельності та практик BDD. Однак, вибираючи його для потреб тестування, враховуйте складність установки та швидкість виконання.

Фреймворк Protractor – це програма Node.js. За замовчуванням Protractor використовує фреймворк тестування Jasmine для свого інтерфейсу тестування[16].

Переваги:

– заснований на Selenium Webdriver: Protractor працює поверх

Selenium Webdriver, що означає, що він успадковує всі функції Webdriver;

– специфічні для Angular функції: Protractor спеціально розроблений для тестування додатків Angular. Він має вбудовану підтримку для ідентифікації специфічних для Angular елементів;

– легке перемикання між додатками Angular та non-Angular;

– паралельне тестування та підтримка кросбраузерності: Це також полегшує кросбраузерне тестування в різних браузерах.

Недоліки:

– крива навчання: Protractor вимагає кривої навчання, особливо для початківців у тестуванні Angular;

– обмежується додатками Angular. Незважаючи на те, що Protractor чудово тестує додатки Angular, він може бути не найкращим вибором для проектів, не пов'язаних з Angular;

– очікування за замовчуванням Angular: чекає, поки Angular стабілізується, перш ніж виконувати дії.

Підводячи підсумок, можна сказати, що транспортир є потужним інструментом для тестування Angular. Для зручності порівняння характеристики фреймворків зведені в таблицю 1.4.

Таблиця 1.4 – Порівняльний аналіз фреймворків інтеграційного тестування веб-додатків [16]

Характеристика (1-5)	JWebUnit	Cucumber	Protractor
Простота налаштування	4	5	5
Простота освоєння	4	4	4
Ком’юніті	3	5	3
Функціональність	3	4	4
Документація	4	4	4
Продуктивність	4	4	5
Підсумок	22	26	25

Як показав аналіз, фреймворк Cucumber має найкращі характеристики для інтеграційного тестування веб-додатків.

1.4.3 E2E-тестування програмного забезпечення

Щоб вибрати фреймворк для тестування E2E, давайте порівняємо характеристики фреймворків Selenium, Puppeteer і Cypress.

Фреймворк Selenium Selenium WebDriver – це цілий набір інструментів, який дозволяє здійснювати автоматизацію браузера. Відмінними рисами Selenium є можливість написання скриптів на JavaScript, C#, Java, Ruby, Python та підтримка більшості сучасних браузерів (Chrome, Firefox, Safari, Edge). Ключовим інструментом для роботи з браузером є Selenium WebDriver.

Сценарії автоматизованого тестування пишуться на одній з бажаних мов, після чого мово-прив'язка Selenium переводить команду в JSON і відправляє її (через HTTP) на сервер Selenium.

За допомогою вбудованих драйверів браузера він обмінюється даними і управляє браузером [17].

Схема обміну даними Selenium WebDriver показана на рисунку 1.8.

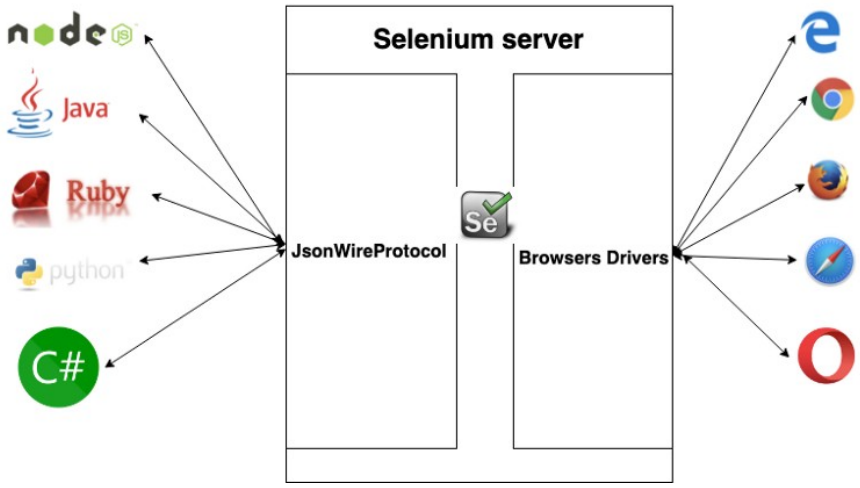

Рисунок 1.8 – Схема обміну даними у Selenium WebDriver

Переваги:

– гнучкість у використанні та виборі мови, платформи, браузера;

– Selenium Webdriver – це де-факто галузевий стандарт, створений на основі затвердженого веб-стандарту W3C WebDriver;

– підтримка паралельного тестового запуску (Selenium Grid);

– підтримка різних плагінів.

Недоліки:

– нетривіальна інсталяція;

– немає вбудованого хорошого інструменту для побудови детального звіту про помилки;

– немає вбудованої можливості порівняння зображень.

Фреймворк Puppeteer – це бібліотека з відкритим вихідним кодом, яка надає високорівневий API для запуску, контролю та управління браузером – Chromium.

На відміну від Selenium Webdriver, зв'язок відбувається безпосередньо з браузером, хоча і за тим же протоколом Chrome DevTools, який використовує Selenium ChromeDriver. Особливість полягає в тому, що Puppereer розвивається і оновлюється набагато динамічніше, ніж це відбувається з ChromeDriver Selenium [16]. Принцип роботи фреймворку Puppeteer показаний на рисунку 1.9.

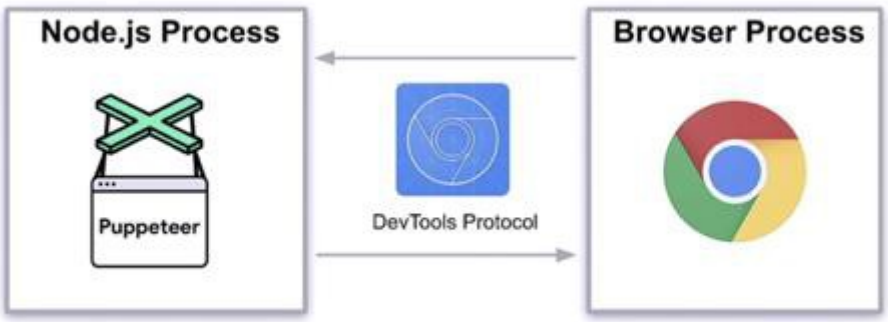

Рисунок 1. 9 – Принцип роботи фреймворку Puppeteer

Переваги:

– багате API для роботи з мережевими запитами та їх перехоплення;

– можливість значно прискорити тести за допомогою Puppeteer Browser Context;

– наявність API для тестування додатку в автономному режимі і т.д.

Недоліки:

– по суті, підтримка роботи тільки з одним браузером - Chromium;

– відсутній вбудований механізм розпаралелювання тестів;

– необхідність підключення додаткових бібліотек.

Фреймворк Cypress – це фреймворк з відкритим вихідним кодом для тестування E2E. Це також відносно молодий інструмент, як і Puppeteer, але він привносить нові концепції та рішення в те, як здійснюється автоматизація та тестування.

Ключова особливість Cypress полягає в тому, що він працює всередині самого браузера. Це означає, що Cypress завжди стежить за викликами всіляких подій в браузері і ніколи не пропустить жодної маніпуляції з елементами сторінки, що значно знижує ймовірність появи плаваючих тестів.

Переваги:

– вбудований набір інструментів для тестування (Mocha, Chai, Sinon)

– вбудований автоматичний резервний механізм;

– функція «Time machine», яка дозволяє зробити відкат до певних кроків у послідовності виконання тесту [15].

Недоліки:

– немає підтримки кросбраузерності (тільки Chomium і Chrome);

– немає можливості створення додаткових вкладок та вікон;

– не призначений для тестування продуктивності тощо.

Для зручності порівняння характеристики фреймворків зведені в таблицю 1.5.

Таблиця 1.5 – Порівняльний аналіз фреймворків для E2E-тестування веб-додатків [16]

Характеристика (1-5)	Selenium WebDriver	Puppeteer	Cypress
Простота налаштування	4	4	4
Простота освоєння	4	4	4
Ком’юніті	5	4	3
Функціональність	5	3	3
Документація	5	4	5
Продуктивність	4	4	4
Підсумок	27	23	23

Як показав аналіз, фреймворк Selenium має найкращі характеристики для E2E-тестування веб-додатків.

З урахуванням вищевикладеного, складена підсумкова таблиця рекомендованих фреймворків за рівнями автоматизації тестування веб-додатків (табл. 1.6).

Таблиця 1.6 – Рекомендовані фреймворки за рівнями автоматизації тестування веб-додатків [16]

Рівень автоматизації тестування	Фреймворк
Модульне тестування	JUnit
Інтеграційне тестування	Cucumber
E2E тестування	Selenium

Обрані фреймворки рекомендовані для використання при розробці моделей ІСАТ ПЗ.

Висновки до розділу:

– проблеми автоматизованого тестування програмного забезпечення за допомогою сучасних інформаційних технологій широко розглядаються в працях вітчизняних і зарубіжних вчених;

– ефективна стратегія автоматизації тестування передбачає автоматизацію тестування на трьох різних рівнях: unit, integration та E2E.

– ефективність роботи команди тестування багато в чому залежить від того, які завдання було вирішено автоматизувати і як ця автоматизація була проведена;

– найефективнішим підходом до автоматизації тестування веб-додатків є використання спеціалізованих фреймворків;

– згідно з концепцією гнучкого проектування Agile, автоматизоване тестування має бути не ізольованим завданням, а безперервним процесом;

– за допомогою порівняльного аналізу за рівнями тестування було відібрано фреймворки, які вважаються найкращими у спільноті розробників.

РОЗДІЛ 2
РОЗРОБКА МОДЕЛЕЙ ТА АЛГОРИТМІВ ІНТЕГРОВАНОГО СЕРЕДОВИЩА АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічне моделювання інтегрованого програмного автоматизованого тестового середовища

Розглянемо процес логічного моделювання ІСАТ для веб-додатків і використаємо методологію RUP (Rational Unified Process).

Методологія RUP – це процес розробки програмного забезпечення для об'єктно-орієнтованих моделей. Вона також відома як єдина модель процесу [14].

На початковому етапі логічного проектування визначаються основні вимоги, обмеження та ключова функціональність продукту. Створено базовий варіант моделі юз-кейс [15].

Мета полягає у визначенні основних функціональних можливостей ІСАТ ПЗ. Ключові учасники – системний аналітик та тестувальник.

Вхідна інформація: список функціональних вимог до ІСАТ ПЗ, список рекомендованих фреймворків за рівнями автоматизації тестування веб-додатків.

Щоб визначити основний функціонал ІСАТ ПЗ, побудуємо діаграму варіантів використання UML.

Діаграми варіантів використання мають ряд переваг при розробці програмного забезпечення та проектуванні систем:

– збірник функціональних вимог. Діаграми прикладів використання допомагають зафіксувати функціональні вимоги системи;

– відстеження. Випадки використання легко відстежити. Кожен випадок використання відповідає певній функціональності або користувацькому досвіду;

– основа для оцінки та планування. Випадки використання забезпечують основу для оцінки, планування та перевірки витрат;

– еволюційного розвитку. Варіанти використання можуть змінюватися протягом усього процесу розробки;

– альтернативні шляхи. Випадки використання дозволяють захоплювати альтернативні шляхи або виняткову поведінку;

– зручна комунікація. Випадки використання легко зрозумілі бізнес-користувачам;

– професійне моделювання. Ви можете створювати професійні моделі варіантів використання за допомогою інструментів моделювання UML.

Акторами процесу тестування є Тестувальник і фреймворки: JUnit, Selenium і Cucumber. Варіанти використання представлені в таблицях 2.1-2.5

Таблиця 2.1 – Опис прецеденту «Авторизація»[10]

Елемент діаграми	Опис
Прецедент	Авторизація
ID	1
Короткий опис	Авторизація користувача
Головний актор	Тестувальник
Другорядний актор	Не має
Передумова	Не має
Основний потік	Тестувальник виконує авторизацію для доступу до ICAT
Післяумова	Не має
Альтернативні потоки	Не має

Таблиця 2.2 - Опис прецеденту «Модульне тестування веб-додатків» [10]

Елемент діаграми	Опис
Прецедент	Модульне тестування веб-додатків
ID	2
Короткий опис	Модульне тестування веб-додатків
Головний актор	Тестувальник

Другорядний актор	Фреймворк JUnit
Передумова	Не має
Основний потік	Тестувальник виконує модульне тестування веб-додатку у ICAT
Післяумова	Не має
Альтернативні потоки	Не має

Таблиця 2.3 – Опис прецеденту «Інтеграційне тестування веб-додатків» [10]

Елемент діаграми	Опис
Прецедент	Інтеграційне тестування веб-додатків
ID	3
Короткий опис	Інтеграційне тестування веб-додатків
Головний актор	Тестувальник
Другорядний актор	Фреймворк Cucumber
Передумова	Не має
Основний потік	Тестувальник виконує інтеграційне тестування веб-додатків в ICAT
Післяумова	Не має
Альтернативні потоки	Не має

Таблиця 2.4 – Опис прецеденту E2E тестування веб-додатків [10]

Елемент діаграми	Опис
Прецедент	E2E тестування ПЗ
ID	4
Короткий опис	E2E тестування ПЗ
Головний актор	Тестувальник
Другорядний актор	Фреймворк Selenium
Передумова	Не має
Основний потік	Тестувальник виконує E2E тестування веб-додатків в ICAT
Післяумова	Не має
Альтернативні потоки	Не має

Таблиця 2.5 – Опис прецеденту «Формування звіту» [10]

Елемент діаграми	Опис
Прецедент	Формування звіту
ID	5
Короткий опис	Формування звіту тестування ПЗ
Головний актор	Тестувальник
Другорядний актор	Не має
Передумова	Не має
Основний потік	Тестувальник формує звіт результатів тестування ПЗ
Післяумова	Не має
Альтернативні потоки	Не має

На рисунку 2.1 представлена діаграма варіантів використання ICAT веб-додатків.

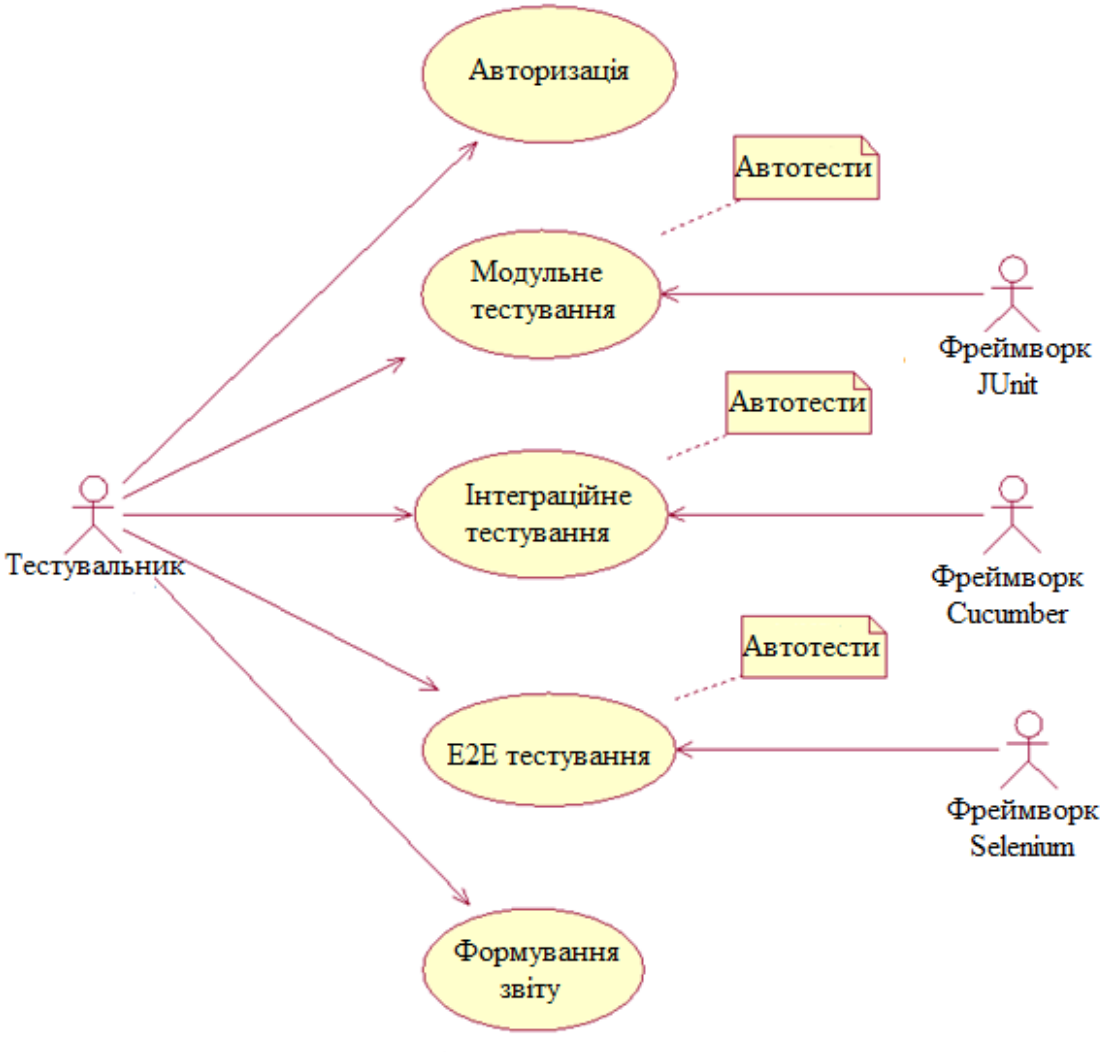

Рисунок 2.1 – Діаграма варіантів використання ICAT веб-додатків

Діаграма побудована з точки зору тестувальника. Розроблена діаграма варіантів використання відображає функціональний аспект і є функціональною моделлю веб-додатків ICAT.

Щоб проаналізувати предметну область автоматизації, розробимо її модель. Модель предметної області (модель бізнес-об'єкта) - це потужний механізм опису важливих бізнес-термінів, що забезпечує єдине визначення термінів і їх взаємозв'язків, доступне для всього персоналу проекту, від бізнес-менеджерів високого рівня до інженерів нижчого рівня.

Однією з переваг використання доменної моделі є те, що терміни моделюються як елементи, що дозволяє пов'язувати їх з іншими елементами в самій моделі предметної області або з елементами в інших частинах моделей [17].

Для розробки моделі предметної області ми використовуємо діаграму класів UML. Діаграма класів є цінним інструментом у розробці програмного забезпечення та проектуванні систем, який дає уявлення про модель даних, огляд системи, комунікацію та розподіл.

На рисунку 2.2 зображена діаграма класів ICAT веб-додатків. В таблиці 2.6 представлена специфікація класів ICAT.

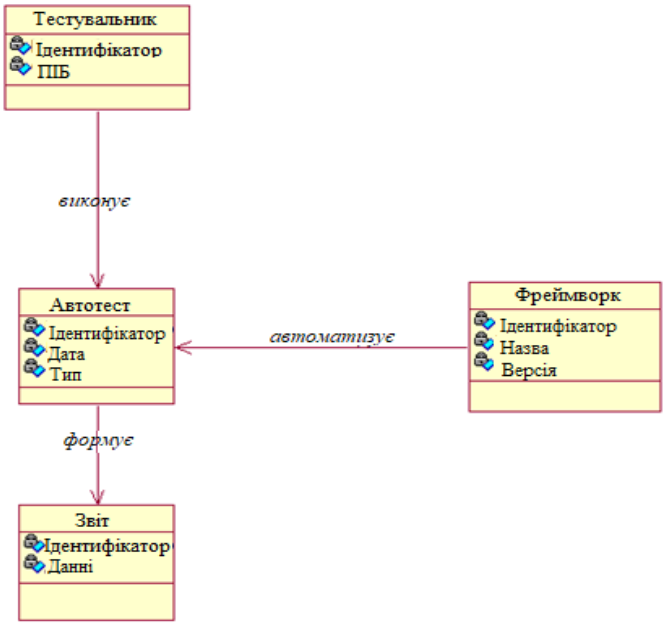

Рисунок 2.2 – Діаграма варіантів використання ICAT веб-додатків

Таблиця 2.6 – Специфікація класів ІС тестування веб-додатків [10]

Клас	Опис
Тестувальник	Клас об'єктів, які представляють на логічному рівні тестувальники програмного забезпечення
Фреймворк	Клас об'єктів, які представляють фреймворки автоматизації тестування на логічному рівні
Автотест	Клас об'єктів, що представляють автоматизовані тести на логічному рівні
Звіт	Клас об'єктів, що представляють на логічному рівні звіти результатів тестування

Для розробки моделі програмної архітектури веб-додатків ICAT використовується схема компонентів UML [19]/

Компонентні діаграми дуже важливі з точки зору реалізації. Ці діаграми широко використовуються для ілюстрації архітектури програмного забезпечення ICAT [17].

Таким чином, команда розробників додатків повинна добре розбиратися в деталях компонентів.

На рисунку 2.3 показана програмна архітектура ICAT веб-додатків, побудована у вигляді діаграми компонентів UML.

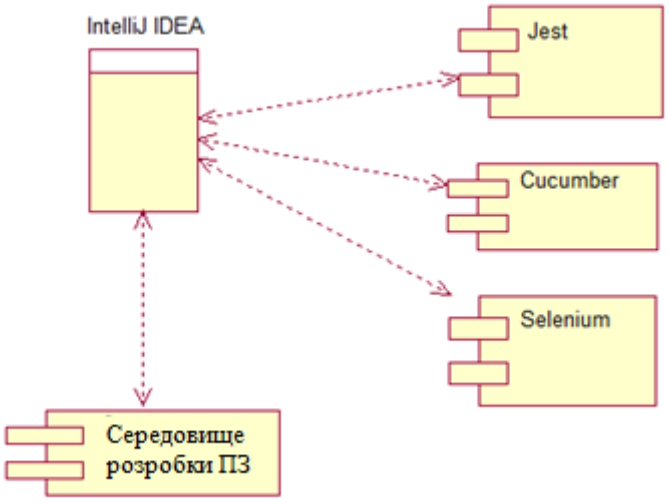

Рисунок 2.3 – Програмна архітектура ICAT веб-додатків

Представлена програмна архітектура відображає загальні залежності між програмними компонентами веб-додатків ICAT.

2.2 Інформаційне моделювання інтегрованого середовища автоматизованого тестування програмного забезпечення

Інформаційна модель – це організаційна структура, яка використовується для категоризації інформаційних ресурсів. Ця структура допомагає творцям і користувачам знаходити те, що їм потрібно, навіть якщо їхні потреби значно відрізняються та є особистими.

Створення інформаційної моделі вимагає аналізу, ретельного планування та великої кількості зворотного зв'язку від вашої спільноти користувачів.

Планування означає комунікацію з широким колом зацікавлених сторін, включаючи окремих осіб і групи, які мають інформаційні потреби і могли б отримати вигоду від співпраці в розробці інформаційних ресурсів.

Щоб отримати зворотний зв'язок, вам потрібно протестувати свою інформаційну модель з членами вашої спільноти користувачів, щоб переконатися, що ви не пропустите деякі важливі точки зору.

Для побудови інформаційної моделі ICAT ПЗ використовувався онлайн-сервіс Visual Paradigm [19].

Інформаційна модель функції тестування програмного забезпечення наведена на рисунку 2.4.

Представлена інформаційна модель базується на моделі бізнес-процесу тестування програмного забезпечення в IT-компанії.

Для розробки бази даних (БД) програмного забезпечення ISAT використовується СУБД MySQL.

Для розробки програмної моделі даних ISAT ми використовуємо інструмент MySQL Workbench CASE, який підтримує стандарт IDEF1X. Workbench надає моделювання даних, розробку SQL та комплексні

інструменти адміністрування для конфігурації сервера, адміністрування користувачів, резервного копіювання тощо.

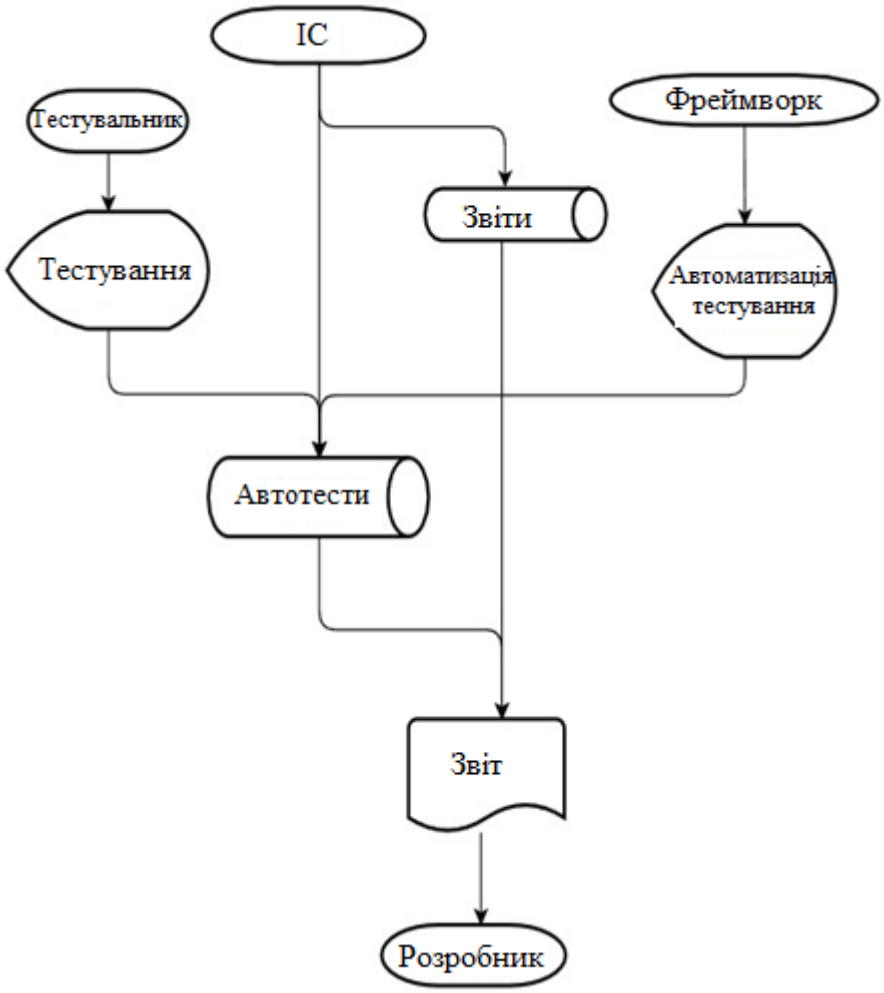

Рисунок 2.4 – Інформаційна модель функції тестування ПЗ

Workbench дозволяє створювати фізичну модель бази даних для MySQL без попереднього логічного моделювання, що дозволяє значно підвищити продуктивність процесу [20].

У процесі розробки діаграми ER-діаграми ІС з її діаграми класів виділяють такі сутності:

– Користувач;
– Автотести;
– Фреймворк;
– Звіт.

На рисунку 2.5 показана модель даних ICAT веб-додатків.

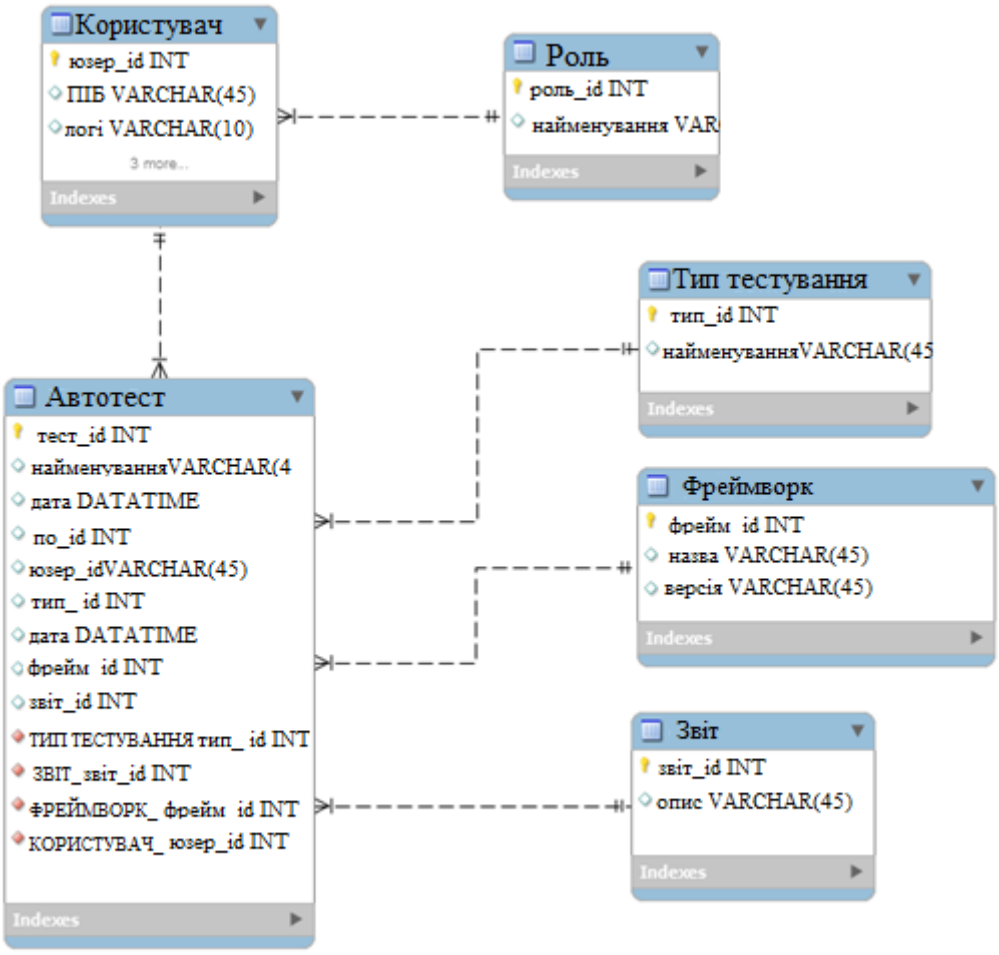

Рисунок 2.5 – Модель даних ICAT веб-додатків

Для приведення даних до третьої форми нормалізації вводяться сутності «Тип Тестування» і «Роль Користувача». Між суб'єктами встановлено такі взаємозв'язки:

– Користувацький автотест – «один до багатьох»;
– Тип тестування - Автоматизований тест - "один до багатьох";
– Перевірене ПЗ-Автотест – «один до багатьох»;
– Фреймворк-Автотест – "один до багатьох";
– Звіт-Автотест – "один до багатьох";
– Роль користувача – «один до багатьох».

Оскільки ICAT веб-додатків належить до систем OLTP, всі відносини між сутностями не ідентифікуються.

2.3 Фізичне проектування інтегрованого середовища автоматизованого тестування програмного забезпечення

Для проектування програмного забезпечення ISAT ми використовуємо підхід, заснований на використанні технологічної платформи, яку ми використовуємо в якості інтегрованого середовища розробки (IDE) або інтегрованого середовища розробки.

IDE – це багатофункціональна програма, яку можна використовувати для різних аспектів розробки програмного забезпечення.

IDE дозволяє програмістам поєднувати різні завдання з написання комп'ютерної програми.

IDE підвищують продуктивність програмістів за рахунок об'єднання загальних дій з написання програмного забезпечення в єдину програму: редагування вихідного коду, створення виконуваних файлів і налагодження.

Типове інтегроване середовище розробки включає:

- текстовий редактор;
- транслятор – компілятор або інтерпретатор;
- засоби автоматизації збірки;
- відладчик [8].

Розглянемо та порівняємо характеристики інтегрованих середовищ розробки, що використовуються для розробки веб-додатків: IntelliJ IDEA, NetBeans, Eclipse.

IntelliJ IDEA є провідним, і на думку багатьох експертів галузі та розробників Java, просто найкращим середовищем швидкої розробки Java на сьогоднішній день.

IntelliJ IDEA – це високотехнологічний набір тісно інтегрованих інструментів програмування, що включає інтелектуальний редактор вихідного коду з просунутими засобами автоматизації, потужні інструменти рефакторингу коду, вбудовану підтримку технологій J2EE, механізми інтеграції з середовищем тестування Ant/JUnit та системами контролю версій,

унікальний інструмент для оптимізації та перевірки коду Code Inspection, а також інноваційний візуальний дизайнер для графічних Інтерфейси (рис.2.6) [30].

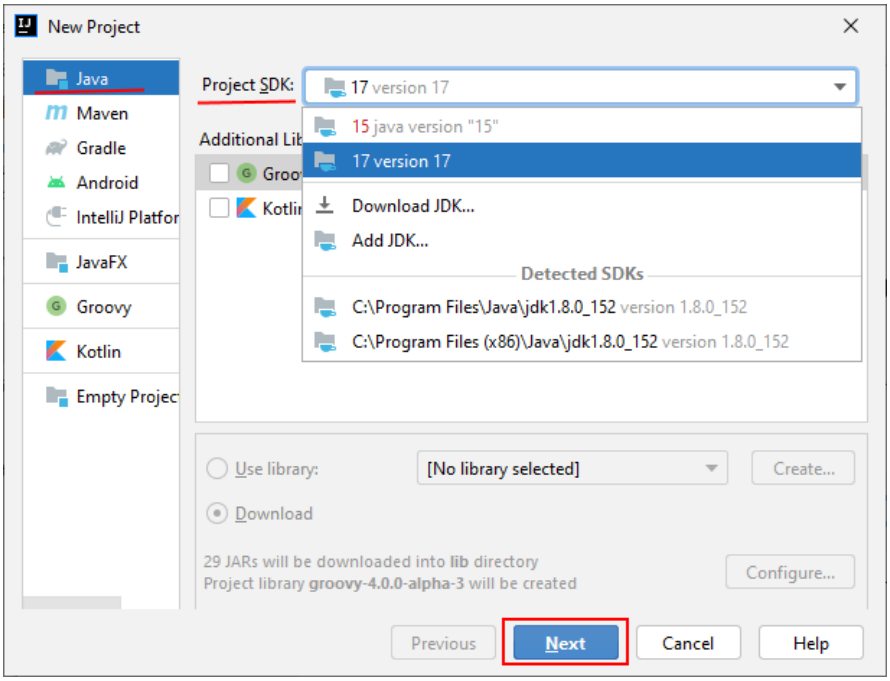

Рисунок 2.6 – Середовище IntelliJ IDEA

Починаючи з версії 9.0, середовище доступне у двох редакціях: Community Edition (безкоштовна) та Ultimate Edition.

Безкоштовна версія підтримує Java, Kotlin, Groovy та Scala; Андроїд; Maven, Gradle і SBT; працює з системами контролю версій Git, SVN, Mercurial та CVS.

NetBeans — це безкоштовне інтегроване середовище розробки з відкритим вихідним кодом для розробників програмного забезпечення (рис.2.7).

NetBeans надає всі інструменти, необхідні для створення професійних настільних, корпоративних, мобільних і веб-додатків на платформі Java, а також C++, PHP та інших мовах.

Середовище NetBeans дозволяє розробляти настільні додатки на Java з професійними графічними призначеними для користувача інтерфейсами, а також створювати веб- і корпоративні додатки відповідно до сучасних

стандартів [20].

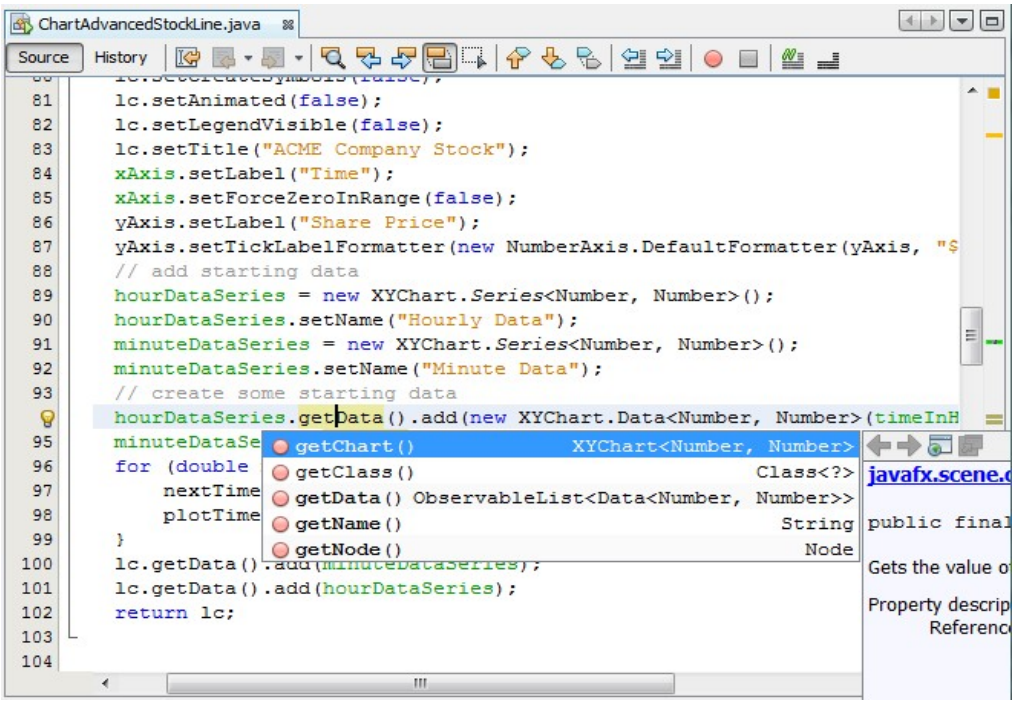

Рисунок 2.7 – Середовище IDE NetBeans

Eclipse IDE — це безкоштовна програмна платформа з відкритим вихідним кодом, яка контролюється Eclipse Foundation.

Середовище написано на мові програмування Java. Основною метою його створення є підвищення продуктивності процесу розробки програмного забезпечення [25]. Фрагмент середовища показаний на рис. 2.8.

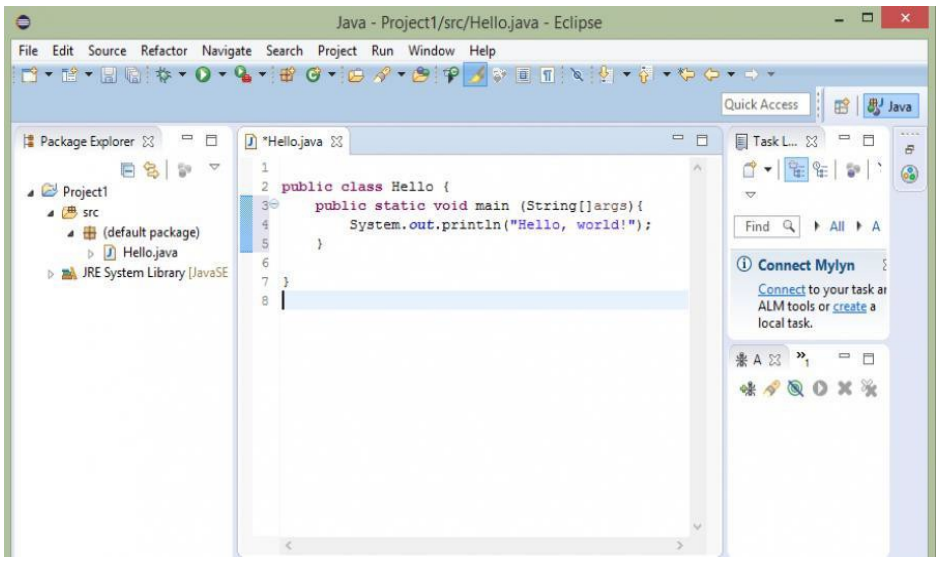

Рисунок 2.8– Середовище IDE Eclipse

Основною складовою є виконуване середовище – Eclipse Runtime, в якому виконується код розширень і модулів.

У ньому передбачений весь базовий функціонал – управління розширеннями та оновленнями, взаємодія з операційною системою, забезпечення роботи довідкової системи.

Наступним ключовим компонентом є саме інтегроване середовище розробки – воно відповідає за управління елементами програми, управління проектами, налагодження та побудову проектів, пошук файлів та командну програму [25].

Для вибору інтегрованого середовища розробки використовуємо таблицю 2.7, складену на основі аналізу блогів на цю тему.

Таблиця 2.7 – Порівняльний аналіз інтегрованих середовищ розробки

Характеристика/бал	IntelliJ IDEA	NetBeans	Eclipse
Юзабіліті	2	2	2
Проста інтеграція з фреймворками автоматизації тестування	3	2	2
Продуктивність	3	2	2
Підсумок	8	6	6

Беручи до уваги результати аналізу, ми обираємо IntelliJ IDEA як середовище для розробки веб-додатків ICAT. Пакетна діаграма надає вигляд модульної конфігурації програмного забезпечення. На рис. 2.9 зображена структурна схема ICAT веб-додатків у вигляді діаграми пакетів.

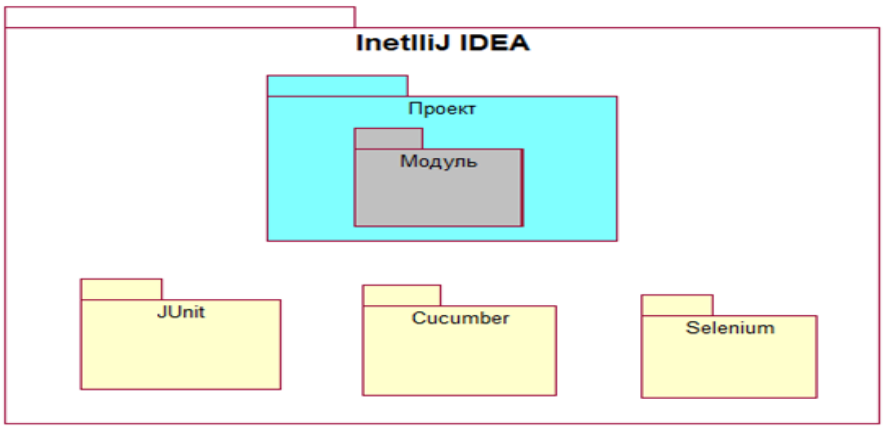

Рисунок 2.9 – Структурна схема ICAT веб-додатків

В таблиці 2.8 дано опис функцій модулів ICAT.

Таблиця 2.8– Опис функцій модулів ICAT [8]

Назва модулю	Функції модулю
IntelliJ IDEA	Надає інструменти для управління процесом тестування веб-додатків
Фреймворк JUnit	Автоматизована підтримка модульного тестування веб-додатків
Фреймворк Cucumber	Автоматизована підтримка тестування інтеграції веб-додатків
Фреймворк Selenium	Автоматизована підтримка E2E-тестування веб-додатків

в IntelliJ складається з модулів (наприклад, модулів java). Модулі містять програмний код.

Таким чином, у концепції IntelliJ модулі – це частини проєкту, які можна компілювати, запускати, тестувати та налагоджувати незалежно один від одного. Модулі – це спосіб знизити трудомісткість великих проектів при збереженні загальної (дизайнерської) конфігурації. Модулі можуть використовуватися повторно: при необхідності модуль може бути включений в більш ніж один проект.

Проект може містити кілька модулів. В даний час основними структурами масштабних проектів є в основному багатомодульні структури.

2.4 Розробка алгоритмів тестування веб-додатків

Діаграма активності зображує потік управління від початкової до кінцевої точки, показуючи різні шляхи прийняття рішень, які існують під час виконання алгоритму.

На рисунку 2.10 показаний алгоритм тестування веб-додатків, побудований у вигляді діаграми активності UML.

Рисунок 2.10 – Алгоритм тестування веб-додатків за рівнями тестування

Існують різні види тестування продуктивності веб-додатків, і одним з них є навантажувальне тестування. Навантажувальні випробування проводяться для визначення працездатності системи в реальних умовах.

Єдина причина провести навантажувальний тест – перевірити, яке навантаження може витримати система. Він підтверджує, чи може пристрій або програмне забезпечення впоратися з навантаженням відповідно до вимог користувача.

Крім того, навантажувальний тест використовується для визначення максимальної працездатності будь-якого додатка. За допомогою навантажувального тестування можна визначити, чи достатня існуюча інфраструктура для запуску програми. Нарешті, тест визначає кількість

користувачів, які можуть працювати над додатком одночасно.

Для навантажувального тестування сайту веб-додатку використовується програма Apache JMeter [21].

Алгоритм навантажувального тестування JMeter показаний у форматі рисунке 2.11.

Рисунок 2.11 – Алгоритм навантажувального тестування веб-додатку за допомогою JMeter

Представлені алгоритми реалізовані в прототипі ІСАТ ПЗ.

2.5 Апробація результатів дослідження

Прототип ІСАТ ПЗ розроблений на базі інтегрованого середовища розробки IntelliJ IDEA. Для представлення ієрархічної структури функцій ІС використовується діаграма дерева функцій, або ієрархічна діаграма.

На рисунку 2.12 показано дерево функцій ІСАТ ПЗ.

Рисунок 2.12 – Дерево функцій ІСАТ ПЗ

IntelliJ IDEA дозволяє налаштовувати параметри на декількох рівнях: на рівні модулів, на рівні проекту та на глобальному.

Глобальні налаштування застосовуються до всіх проєктів, які відкриваються за допомогою певної інсталяції або версії IntelliJ IDEA.

До таких налаштувань можна віднести зовнішній вигляд оточення (теми, колірні схеми, меню та панелі інструментів), налаштування сповіщень, набір встановлених та увімкнених плагінів, налаштування відладчика, доопрацювання коду тощо.

Щоб налаштувати середовище, виберіть пункт меню Файл|

Налаштування для Windows і Linux» (рисунок 2.13).

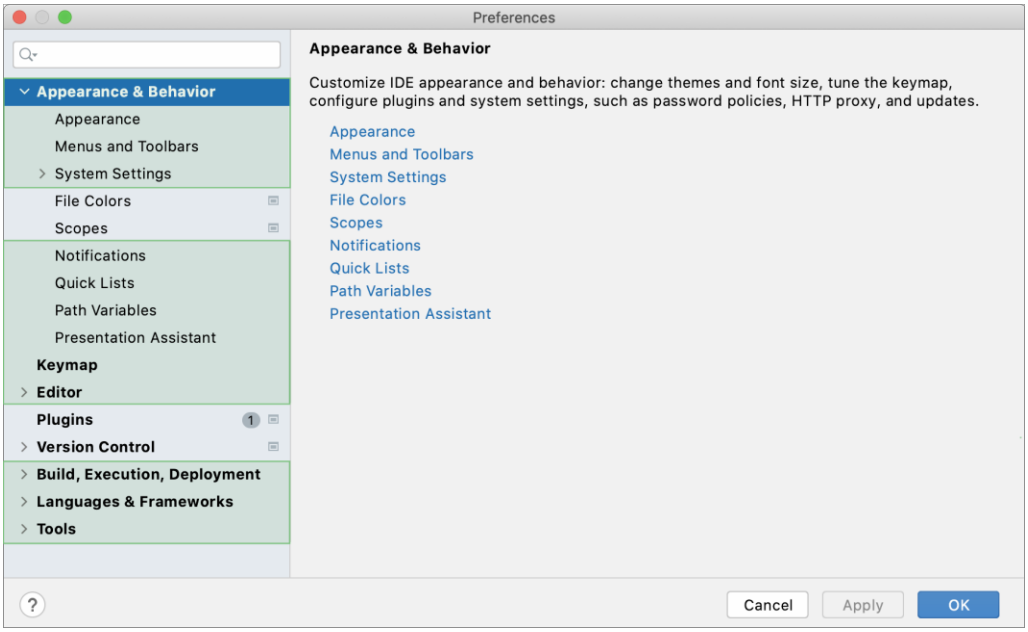

Рисунок 2.13 – Вікно налаштувань конфігурації ICAT ПЗ

Налаштування, не позначені піктограмою «Для поточного проекту» в діалоговому вікні «Settings/Preferences», є глобальними та застосовуються до всіх існуючих проектів у поточній версії IntelliJ.

Інтеграція фреймворків в ICAT здійснюється через опцію «Плагіни», останні версії яких можна завантажити з сайтів розробників. Процес реалізації прототипу веб-додатків ICAT – це інтеграція середовища розробки IntelliJ IDEA з фреймворками JUnit, Cucumber та Selenium.

Бібліотека для JUnit поставляється з IntelliJ IDEA, але за замовчуванням не включена в шлях до класу проекту або модуля.

Тому при створенні тестового класу не допускаються посилання на клас TestCase або текстові анотації.

Щоб додати потрібну бібліотеку до шляху до класу, можна використовувати загальну процедуру додавання залежності до модуля.

Відповідні функції доступні при створенні тесту для класу або при написанні коду для тесту з підменю "Testing-JUnit" (рисунок 2.14) [24].

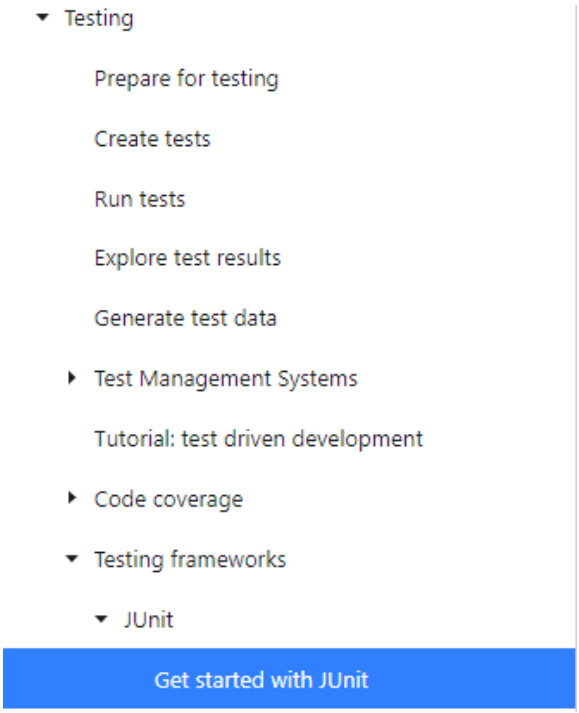

Рисунок 2.14 – Вікно підменю JUnit-тестування

Щоб мати можливість використовувати фреймворк Cucumber у своєму додатку, вам потрібно переконатися, що необхідні плагіни включені, і додати бібліотеку Cucumber до свого проекту.

В IntelliJ Ultimate необхідні плагіни об'єднані та включені за замовчуванням. Однак розробники рекомендують переконатися, що вони включені. У спільноті IntelliJ потрібні вам плагіни не пов'язані між собою, тому їх потрібно встановити та активувати.

Для цього в розділі Settings/Preferences (Ctrl + Alt + S) обираємо Plugins.

Перейдіть на вкладку Installed та переконайтеся, що в вказаному порядку ввімкнено такі плагіни:

- Gherkin;
- Cucumber for Java;
- Cucumber for Groovy (не обов’язково).

Якщо плагіни не встановлені, перейдіть у вкладку Marketplace, введіть їх назви в поле пошуку в зазначеному порядку і натисніть на кнопку Install поруч з кожним з них.

Збережіть зміни та закрийте діалогове вікно. Якщо з'явиться відповідний запит, перезапустіть IDE (рисунок 2.15).

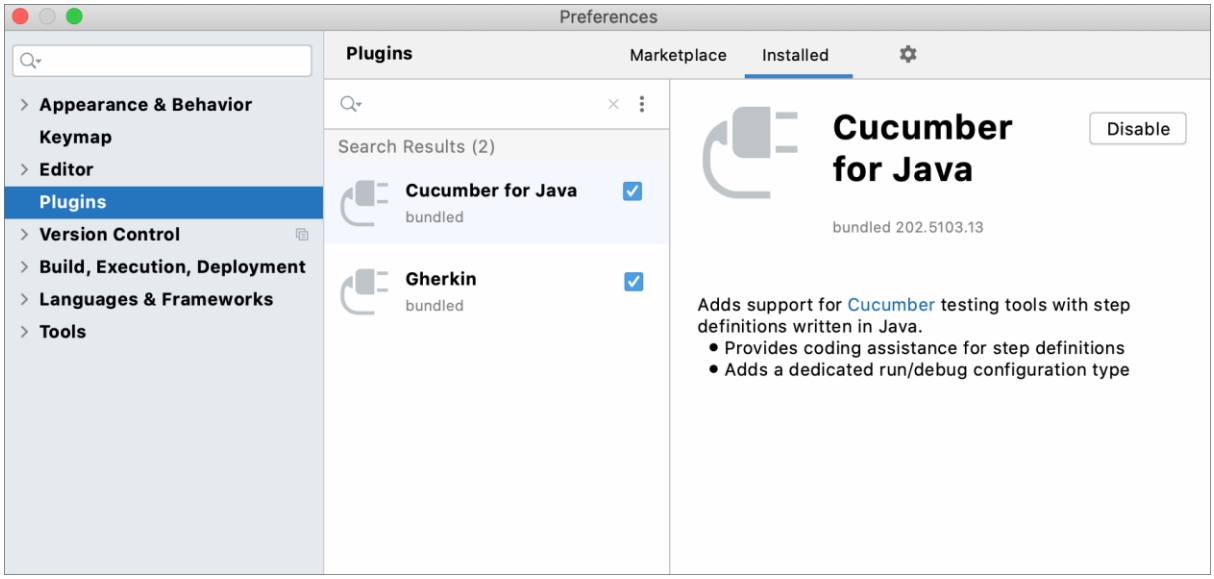

Рисунок 2.15 – Вікно установки плагінів Cucumber

Далі необхідно підключити бібліотеку Cucumber (рисунок 2.16).

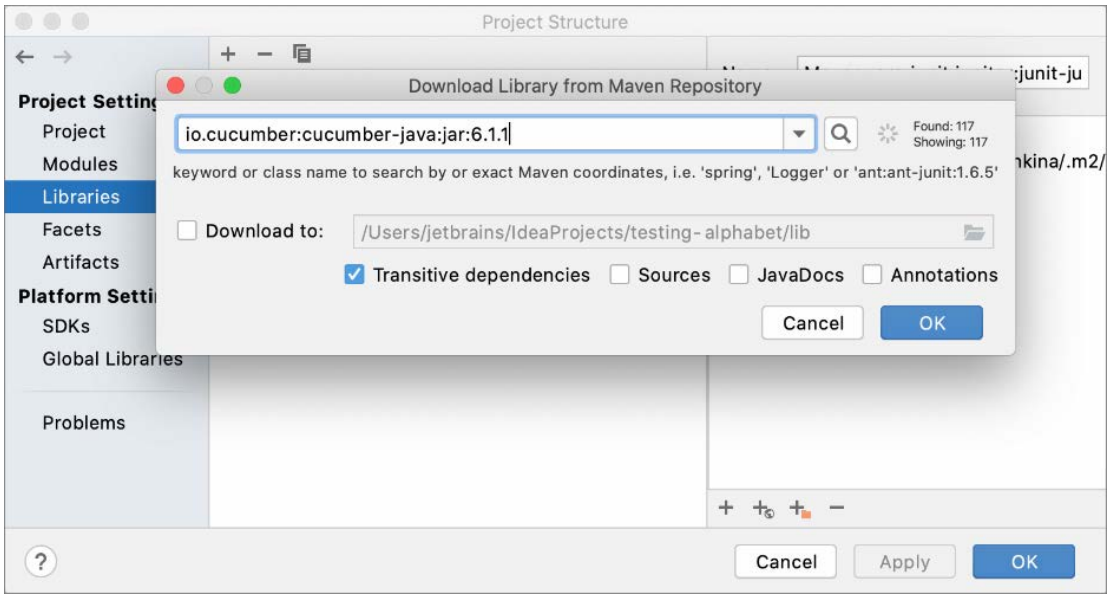

Рисунок 2.16 – Вікно підключення бібліотеки Cucumber

Щоб використовувати фреймворк Selenium, потрібно встановити його плагін.

IntelliJ IDEA 2020.1 Ultimate представляє початкову підтримку Selenium за допомогою нового плагіна Selenium UI Automation Testing.

Новий плагін підтримує найпопулярніші платформи JVM для тестування інтерфейсу та бібліотеки звітів: Selenium, Selenide, Geb, Serenity BDD та Allure Framework.

Щоб встановити цей плагін, виберіть Плагіни в діалоговому вікні Settings/Preferences (Ctrl + Alt + S). Перейдіть на вкладку Marketplace, введіть Selenium UI Testing і натисніть на кнопку Встановити.

Збережіть зміни та закрийте діалогове вікно. Якщо з'явиться відповідний запит, перезапустіть IDE (рисунок 2.17).

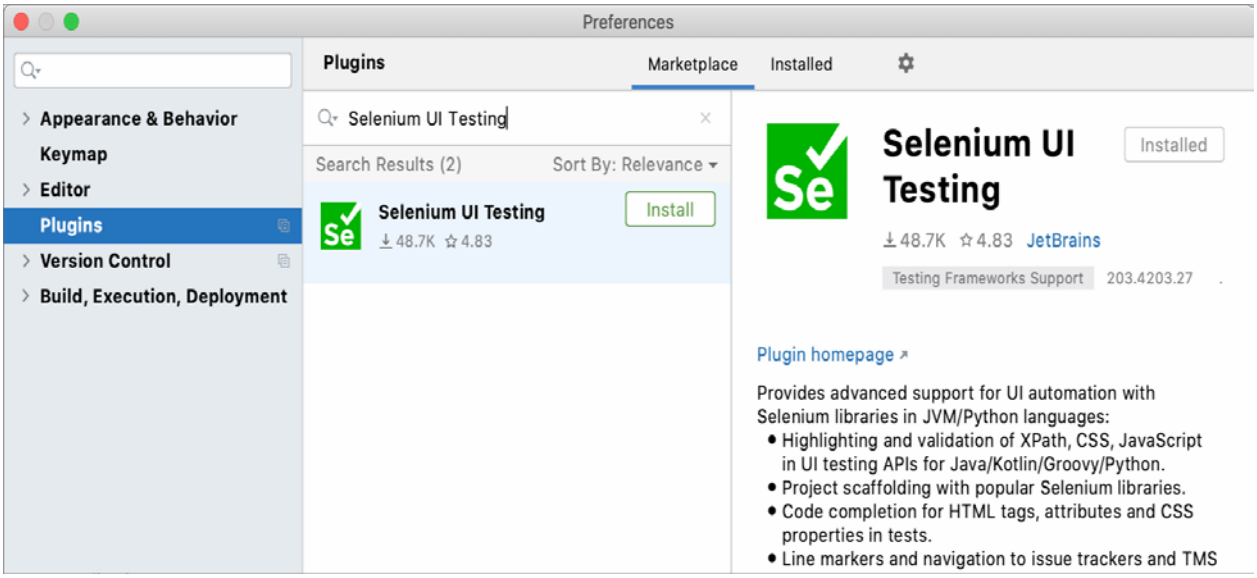

Рисунок 2.17 – Вікно установки плагінів Selenium UI Testing

Створений прототип ІСАТ ПЗ є інтегрованим на базі платформи IntelliJ IDEA.

Для перевірки результатів дослідження було проведено автоматизоване тестування веб-додатку в розробленому ІСАТ

Давайте розглянемо, як реалізувати можливості тестових типів веб-додатків.

Модульне тестування веб-додатку проводиться за допомогою фреймворку JUnit.

Для початку вам потрібно створити проект JUnit (рисунок 2.18).

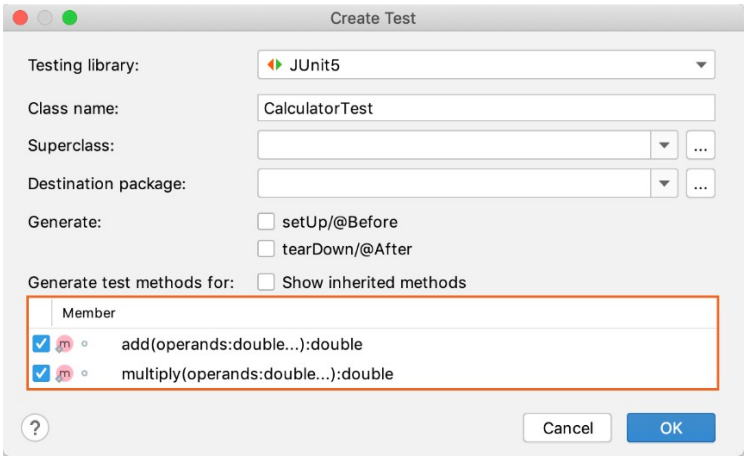

Рисунок 2.18 – Вікно створення тесту JUnit

Після того, як ви налаштуєте код для тестування, ви можете запустити тести і з'ясувати, чи коректно працюють перевірені методи.

Щоб запустити індивідуальне тестування, натисніть на іконку конфігурації запуску тесту в полі та виберіть "Виконати" (рисунок 2.19).

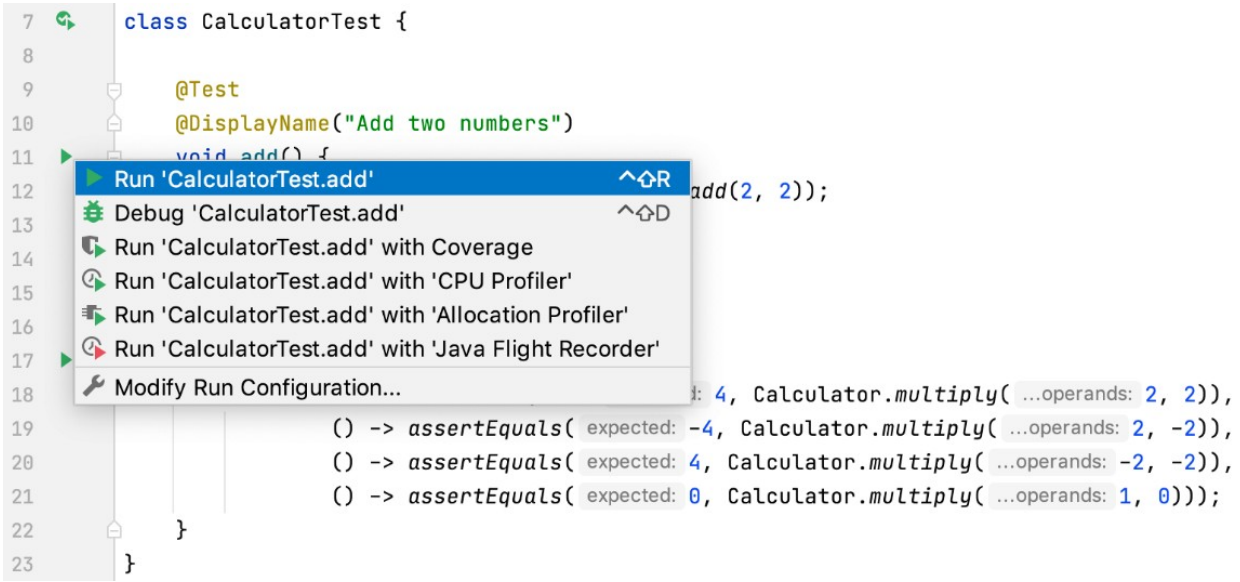

Рисунок 2.19 – Вікно запуску JUnit-тесту

Ви можете переглянути результати тестування безпосередньо у вікні виконання тесту (рисунок 2.20).

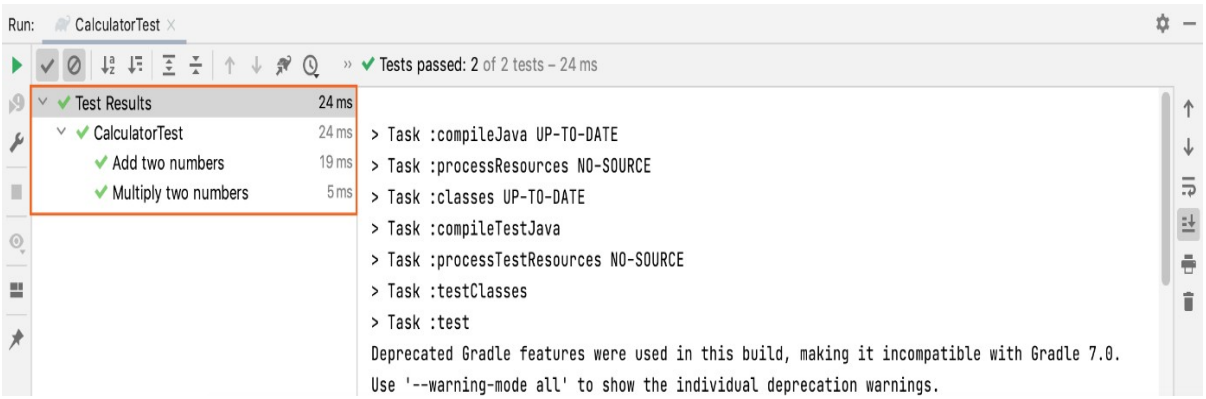

Рисунок 2.20 – Вікно перегляду результату виконання JUnit-тесту

Інтеграційне тестування веб-додатку проводиться за допомогою фреймворку Cucumber.

По-перше, вам потрібно створити структуру папок для проекту Cucumber, як показано на рисунку 2.21.

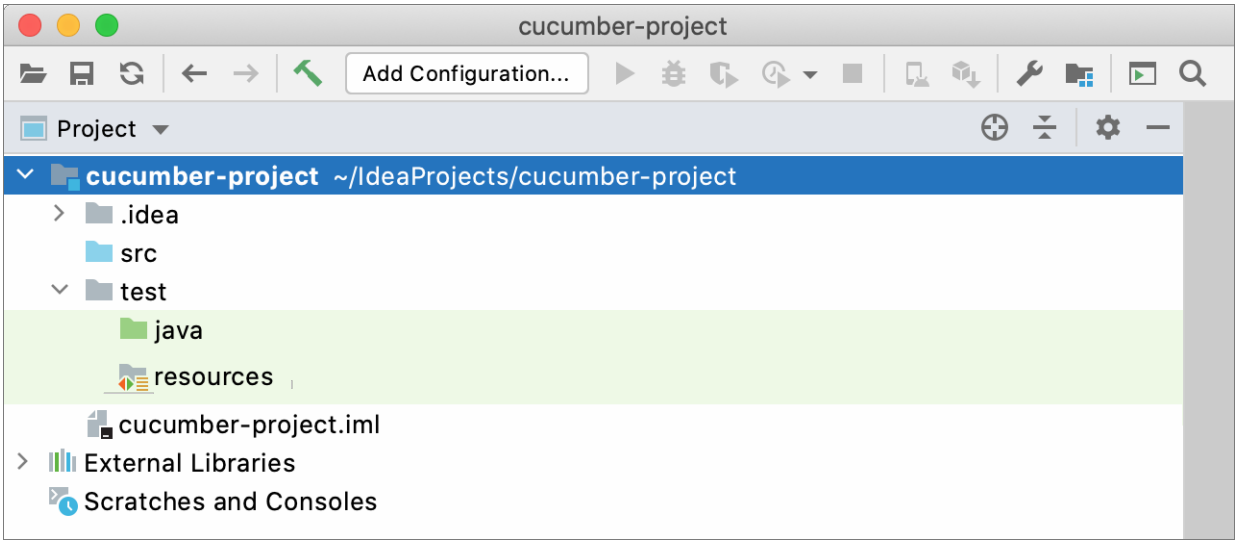

Рисунок 2.21 – Структура папок Cucumber-проекту

Найшвидший спосіб запустити тести Cucumber – використовувати іконки в полі поруч із функцією або скриптом, який ви хочете запустити. Іконки змінюються в залежності від статусу тесту.

На рисунку 2.22 показано вікно виконання тестового скрипту. Для цього потрібно активувати опцію "Запустити тест" в поле поруч зі скриптом, який ви хочете запустити, і вибрати "Виконати" "Сценарій: <ім'я>".

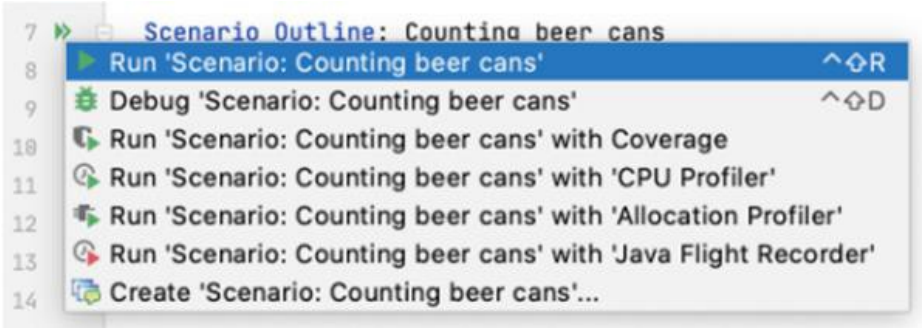

Рисунок 2.22 – Вікно виконання тестового сценарію

Коли ви завершуєте виконання тестів, результати відображаються на вкладці Run Tests у вікні інструмента Run (рисунок 2.23).

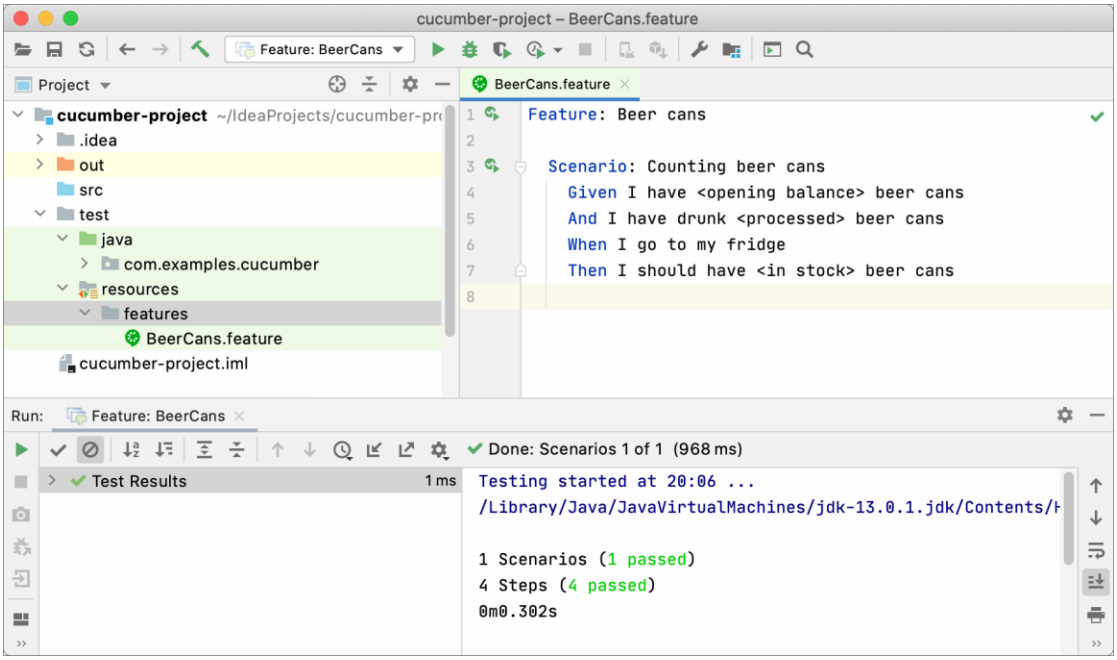

Рисунок 2.23 – Вікно для відображення результатів тестування за допомогою фреймворку Cucumber

З цієї вкладки можна перезапустити тести, експортувати та імпортувати результати тестів, подивитися, скільки часу зайняло виконання кожного тесту і так далі.

E2E-тестування веб-додатку проводиться за допомогою фреймворку Selenium. Створіть новий проект Selenium, як показано в рисунку 2.24.

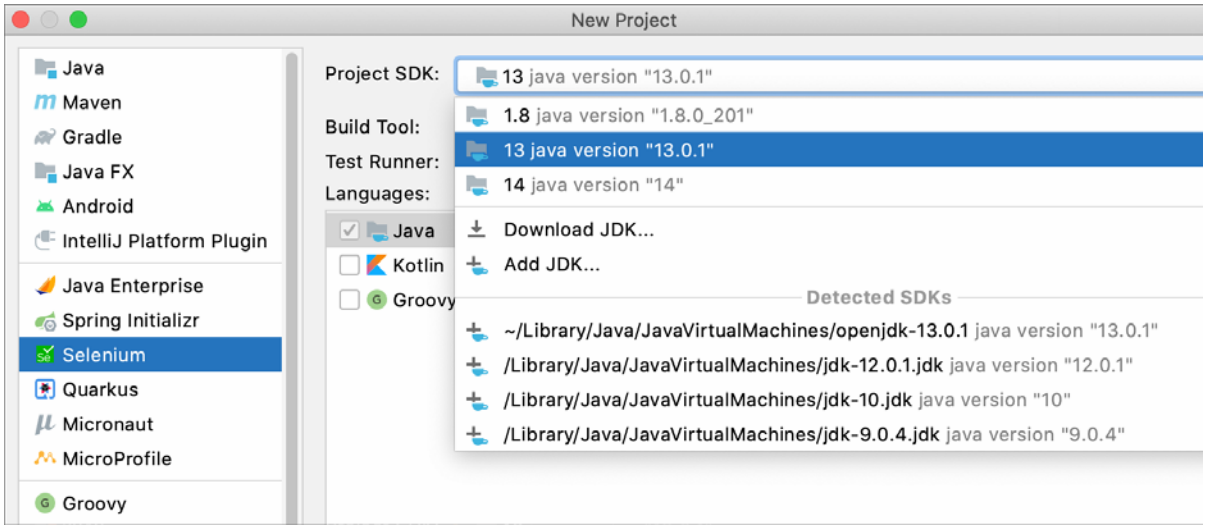

Рисунок 2.24– Вікно створення Selenium-проекту

Плагін Selenium UI Testing надає допомогу в кодуванні для XPath і CSS, які використовуються в Selenium API, а також багатьох інших бібліотеках для тестування UI.

Це включає виділення синтаксичних помилок у селекторах під час введення тексту та доповнення коду для стандартних елементів CSS/HTML (рисунок 2.25).

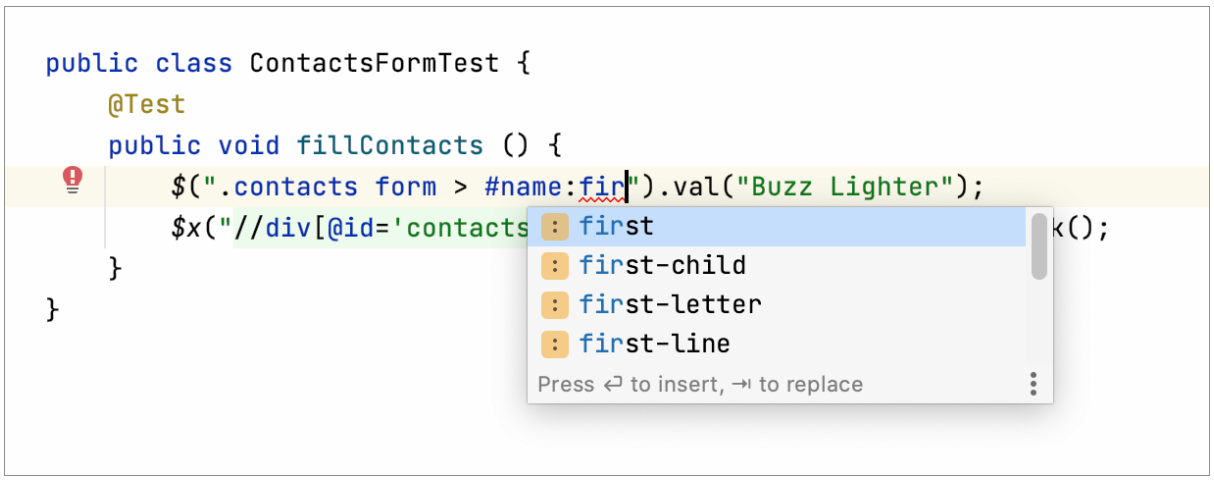

Рисунок 2.25– Вікно створення CSS/HTML-тесту

Плагін Selenium UI Testing надає інструмент на основі Chromium, який дозволяє відкривати веб-сторінки та генерувати код, сумісний із шаблоном об'єкта сторінки, безпосередньо в IDE.

Це скорочує час, витрачений на шаблонний код, і дозволяє зосередитися

на логіці тестування.

Щоб запустити тест, ви повинні навести курсор на клас тесту, щоб виконати всі тести в цьому класі, або на метод тестування, і використовувати комбінацію Ctrl+Shift+F10.

Ви також можете натиснути піктограму кнопки «Запустити» поруч із класом тестування або методом тестування (рисунок 2.26).

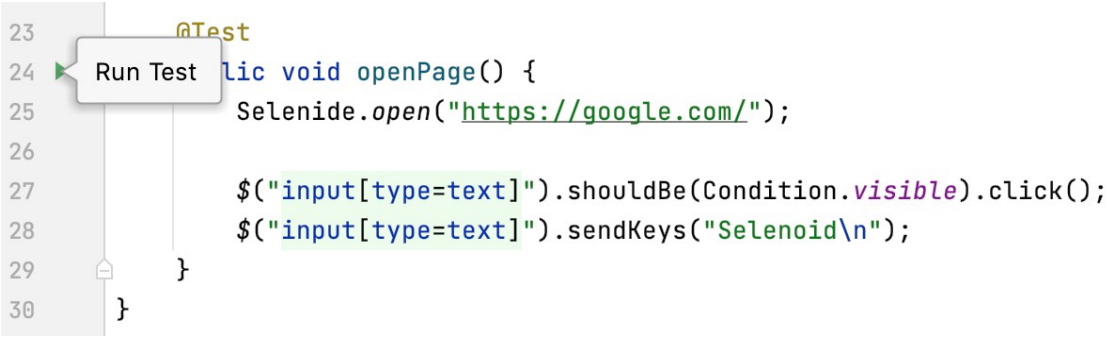

Рисунок 2.26 – Запуск Selenium-тесту

Коли тести завершені, результати відображаються на вкладці «Виконання тесту» у вікні інструменту «Виконати».

З цієї вкладки можна перезапустити тести, експортувати та імпортувати результати тестів, подивитися, скільки часу зайняло виконання кожного тесту і так далі.

Апробація підтвердила працездатність запропонованого програмного рішення ICAT на всіх рівнях автоматизованого тестування веб-додатків.

Висновки до розділу:

– на початковому етапі логічного проектування визначаються основні вимоги, обмеження та ключова функціональність продукту;

– для проектування програмного забезпечення ICAT був використаний підхід, заснований на використанні технологічної платформи, яка використовується в якості IDE або інтегрованого середовища розробки;

– за результатами аналізу, IntelliJ IDEA було обрано як середовище для розробки веб-додатків компанією ICAT;

– діаграма діяльності зображує потік управління від початкової до кінцевої точки, показуючи різні шляхи прийняття рішень, які існують під час виконання алгоритму;

– навантажувальний тест використовується для визначення максимальної працездатності будь-якої програми. Для навантажувального тестування сайту веб-додатку використовується програма Apache JMeter.

– представлені алгоритми реалізовані в прототипі програмного забезпечення ICAT. Прототип ICAT ПЗ розроблений на базі інтегрованого середовища розробки IntelliJ IDEA. Для представлення ієрархічної структури функцій ICAT була побудована діаграма дерева функцій;

– процес реалізації прототипу програмного забезпечення ICAT – це інтеграція середовища розробки IntelliJ IDEA з фреймворками JUnit, Cucumber та Selenium;

– інтеграція фреймворків в ICAT здійснюється через опцію «Плагіни», останні версії яких можна завантажити з сайтів розробників;

– апробація підтвердила працездатність запропонованого програмного рішення ICAT на всіх рівнях автоматизованого тестування веб-додатків;

Таким чином, проведені апробації та розрахунки підтвердили працездатність та високу функціональну ефективність управління програмним забезпеченням ICAT.

ВИСНОВКИ

Робота присвячена актуальній проблемі розробки моделей та алгоритмів для інтегрованого середовища автоматизованого тестування програмного забезпечення (ІСАТ). У процесі її реалізації були вирішені наступні задачі:

– Проведено аналіз стану наукових досліджень і розробок у галузі побудови інтегрованих середовищ для автоматизованого тестування програмного забезпечення. Як показав аналіз, проблеми автоматизованого тестування програмного забезпечення за допомогою сучасних інформаційних технологій широко розглядаються в працях вітчизняних і зарубіжних вчених. Ефективна стратегія автоматизації тестування передбачає автоматизацію тестування на трьох різних рівнях: unit, integration та E2. Ефективність роботи команди тестування багато в чому залежить від того, які завдання було вирішено автоматизувати і як ця автоматизація проводилася. Найефективнішим підходом до автоматизації тестування веб-додатків є використання спеціалізованих фреймворків. Необхідно констатувати недостатність робіт, присвячених розробці моделей та алгоритмів для інтегрованого середовища автоматизованого тестування програмного забезпечення, що підтверджує актуальність теми дослідження;

– Проведено аналіз методик побудови інтегрованих середовищ автоматизованого тестування програмного забезпечення. Як показав аналіз, згідно з концепцією Agile дизайну, автоматизоване тестування має бути не ізольованим завданням, а безперервним процесом, який за своєю суттю вписаний у життєвий цикл програмного забезпечення. За допомогою порівняльного аналізу за рівнями тестування було відібрано фреймворки, які вважаються найкращими у спільноті розробників. Обрані фреймворки рекомендуються для використання при розробці програмних моделей ІСАТ;

– Розроблено моделі та алгоритми програмного забезпечення ІСАТ. На початковому етапі логічного проектування визначаються основні вимоги, обмеження та ключові функціональні можливості продукту, а також

створюється базова версія моделі сценарію використання. Для проектування програмного забезпечення ICAT використовується підхід, заснований на використанні технологічної платформи, яка є IDE або інтегрованим середовищем розробки. Беручи до уваги результати аналізу, IntelliJ IDEA було обрано середовищем для розробки веб-додатків ICAT. Розроблено алгоритми тестування веб-додатків;

– Були проведені апробація та оцінка ефективності проектних рішень. Прототип ICAT ПЗ розроблений на базі інтегрованого середовища розробки IntelliJ IDEA. Для представлення ієрархічної структури функцій ICAT була побудована діаграма дерева функцій;

– Процес реалізації прототипу програмного забезпечення ICAT – це інтеграція середовища розробки IntelliJ IDEA з фреймворками JUnit, Cucumber та Selenium. Інтеграція фреймворків в ICAT здійснюється через опцію «Плагіни», останні версії яких можна завантажити з сайтів розробників.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1 Roger Pressman, Bruce Maxim. Software Engineering: A Practitioner's Approach, 9th Edition. Published by McGraw-Hill Education, 2022. 977 pp.

2 Nina S. Godbole. Software Quality Assurance: Principles and Practices for the new Paradigm. Published by Alpha Science International, 2017. 1066 pp.

3 Boris Beizer. Software Testing Techniques. Published by Van Nostrand Reinhold, 2007, 550 pp.

4 Молодцова О.П. Управління якістю програмної продукції: навчальний посібник. К : КНЕУ, 2001. 248 с.

5 . Cem Kaner et al. Testing Computer Software, 2nd ed. International Thompson Computer Press, 1999. 496 p

6 . Dorothy Graham, Rex Black, Erik van Veenendaal. Foundations of Software Testing ISTQB Certification. Engage Learning, 2019. 243 p.

7 . Horch, John W. Practical guide to software quality management. Artech House, 1996. 259 p.

8 Коробєйник А.Н. Основи тестування ПЗ. К : 2012. 126 с.

9 Laiza Crispin, Janet Gregori. Agile Testing: A Practical Guide for Testers and Agile Teams. Addison-Wesley Signature Series: 2010. 464 p.

10 Сайт Sahi Pro. url: https://www.sahipro.com (дата звернення: 10.03.2025).

11 Сайт Selenium IDE. url: https://www.selenium.dev/selenium-ide (дата звернення: 10.03.2025).

12 Сайт Watir. url: http://watir.com (дата звернення: 10.03.2025).

13 Apache JMeter. url: https://jmeter.apache.org (дата звернення: 30.03.2025).

14 Apache NetBeans. url: https://netbeans.apache.org (дата звернення: 30.03.2025).

15 Automated Testing Strategy: How to Build & Examples. url: https://performancelabus.com/automated-testing-strategy-how-to- build-examples/

#2 (дата звернення: 30.03.2025).

16 Cohn M. The Forgotten Layer of the Test Automation Pyramid. url: https://www.mountaingoatsoftware.com/blog/the- forgotten-layer-of-the-test-automation-pyramid (дата звернення: 25.04.2025).

17 Create a Domain Model. url: https://sparxsystems.com/enterprise_ architect_user_guide/15.2/model_domains/cre ate_a_domain_model.html (дата звернення: 25.04.2025).

18 Eclipse IDE.URL: https://www.eclipse.org/eclipseide (дата звернення: 25.04.2025).

19 Fowler M. Test pyramid. url: https://martinfowler.com/bliki/TestPyramid.html (дата звернення: 25.04.2025).

20 Garcıa B, Duenas J.C. “Automated Functional Testing based on the Navigation of Web Applications”, EPTCS 61, 2011, pp. 49-65.

21 Girgis M.R. et al. “An Automated Web Application Testing System”, International Journal of Computer Applications, vol. 99(7). 2014, pp. 37-44.

22 Hanna M. et al. “Automated Software Testing Framework for Web Applications”, International Journal of Applied Engineering Research, vol. 13(11). 2018, pp. 9758-9767.

23 JetBrains IntelliJ IDEA. url: https://www.jetbrains.com/ (дата звернення: 15.04.2025)

24 JUnit 5. URL: https://junit.org/junit5/ (дата звернення: 15.04.2025)

25 Kalmo M. “Automated Testing of Java Web Applications”, Department of Computer Science and Engineering, University of Göteborg. 2009, 48 p.

26 MySQL Workbench. url: http://www.mysql.com/products/ workbench/features.html (дата звернення: 15.04.2025).

27 The component diagram. url: https://developer.ibm.com/articles/the-component-diagram (дата звернення: 15.04.2025).

28 Types of Testing Environments. url: https://www.testenviron mentmanagement.com/types-of-testing-environments (дата звернення:

15.04.2025)

29 Visual Paradigm Online. url: https://online.visual-paradigm.com (дата звернення: 15.04.2025)

30 Моркун Н. В., Завсегдашня І. В. Методичні вказівки до виконання кваліфікаційної роботи бакалавру для студентів спеціальності 122 “Комп’ютерні науки”. Кривий Ріг : Видавничий центр КНУ, 2021. 50 с.

31 ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура і правила оформлення. Київ, ДП «УкрННЦ», 2015. 26с. (Інформація та документація).

32 ДСТУ 8302:2015. Бібліографічне посилання. Загальні вимоги та правила складання Київ, ДП «УкрННЦ», 2016. 16 с. (Інформація та документація).

33 ДСТУ 3582:2013. Бібліографічний опис. Скорочення слів і словосполучень в українській мові. Загальні вимоги та правила. Київ, ДП «УкрННЦ», 2013. 23 с. (Інформація та документація).