УДК 004.415 #
DOI: https://doi.org/10.17721/1812-5409.2025/2.26

Даниїл ЛІПСЬКИЙ, асп.
ORCID ID: 0009-0000-4068-9453
e-mail: lipsky@knu.ua
Київський національний університет імені Тараса Шевченка, Київ, Україна

АРХІТЕКТУРА ТА ФУНКЦІОНАЛЬНІ ОСОБЛИВОСТІ ПЛАТФОРМИ ДЛЯ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ВЕБЗАСТОСУНКІВ

Представлено платформу для автоматизованого тестування вебзастосунків на основі об'єктно-орієнтованого підходу із застосуванням C#. Платформа призначена для масштабованого та гнучкого тестування вебзастосунків, підтримуючи можливості UI- та API-тестування. Описано основні компоненти системи, що проаналізовані з позиції їхньої функціональності та ролі в автоматизації. Детально розглянуто структуру та взаємодію компонентів, що забезпечують ефективне управління браузерним драйвером, налаштування тестового середовища, збір й обробку тестових даних. Особливістю є використання патерну "singleton" для роботи із браузерним драйвером і підтримка кросбраузерного тестування, що дає змогу перевіряти вебзастосунки на різних платформах. Платформа також підтримує динамічне налаштування та розширення можливостей завдяки конфігураційним файлам і централізованому управлінню параметрами. Система забезпечує зручний пошук вебелементів та їхніх колекцій і розширює функціональність для інтеграційного тестування API та роботи з локальним сховищем браузера. Платформа орієнтована на автоматизацію як сценарного, так й утилітарного тестування, сприяючи ефективному підходу до перевірки функціональності та стабільності вебзастосунків.

Ключові слова: ***автоматизоване тестування, вебзастосунки, C#, кросбраузерне тестування, конфігурація середовища, UI-тестування, API-тестування.***

Класифікація відповідно до AMS 2020: 68M15, 68N19.

Вступ

В умовах сучасного швидкого розвитку вебтехнологій автоматизація тестування вебзастосунків стає ключовим фактором для забезпечення якості програмного забезпечення. Ручне тестування вже не здатне повністю відповідати вимогам ефективності, надійності та швидкості виведення продукту на ринок. Автоматизація дає змогу швидше виявляти помилки, охоплювати широке коло тестових сценаріїв і значно скорочувати час, необхідний для регресійних перевірок.

Традиційне тестування вебзастосунків передбачає перевірку функціональності інтерфейсу користувача, поведінки сторінок та їхніх компонентів у різних браузерах і середовищах. Однак, крім інтерфейсного тестування, важливо також проводити тестування API для забезпечення коректного обміну даними між фронтендом і бекендом. Комплексний підхід до автоматизованого тестування, що охоплює як UI-, так і API-тести, дає змогу ефективно перевіряти всі рівні вебзастосунку, що є особливо важливим для складних багатошарових систем (Forgacs, & Kovacs, 2024).

Метою представленої статті є опис запропонованої автором платформи у вигляді бібліотеки для автоматизації тестування вебзастосунків, що об'єднує можливості UI- та API-тестування, підтримує кросбраузерність і надає інструменти для детальної валідації й генерації звітів. В основі платформи використано сучасні інструменти і технології, що забезпечують її легке масштабування й адаптацію під вимоги конкретного проєкту.

1. Архітектура платформи

Платформа для автоматизованого тестування є комплексним інструментом, що забезпечує точне, стабільне та гнучке виконання тестів. Її архітектура побудована так, щоб інтегрувати всі необхідні компоненти в єдиний потік виконання, охоплюючи такі ключові аспекти, як робота з вебдрайвером, обробка елементів, верифікація, логування, генерація звітів та багато ін. Завдяки запропонованій структурі, платформа дає змогу виконувати тестування навіть у найскладніших сценаріях, забезпечуючи масштабованість, простоту у використанні та високу ефективність. Наведемо опис основних модулів, з яких складається система.

Модуль WebDriverManager забезпечує запуск і контроль браузера під час виконання тестів, а його основними класами є IWebDriverManager і WebDriverManager (Homes, 2024). Інтерфейс IWebDriverManager задає контракт для роботи з драйвером. Він охоплює властивості та методи для отримання екземпляру вебдрайвера, використання механізмів очікування (WebDriverWait), пошуку елементів на сторінках (ElementFinder, ElementsFinder) та ведення логів через LogWorker. Метод OpenApplicationStartPage відповідає за відкриття стартової сторінки тестованого вебзастосунку.

Клас WebDriverManager реалізує цей інтерфейс і відповідає за створення драйвера відповідно до конфігурацій. При ініціалізації він отримує об'єкт TestSettings, де зберігаються всі параметри, включаючи тип браузера, версію, режим виконання та URL вебзастосунку. Локальний запуск драйвера здійснюється через метод GetWebDriver. У ньому враховують вибраний браузер (Chrome, Firefox, Safari), і застосовують відповідні налаштування. Для віддаленого виконання використовують метод GetRemoteWebDriver, що створює драйвер на основі Grid URI, заданого в конфігурації.

WebDriverWait реалізується через властивість Lazy<WebDriverWait> для ефективного управління очікуваннями. Це дає змогу виконувати динамічні дії з елементами без зайвого навантаження.

Метод OpenApplicationStartPage здійснює масштабування вікна браузера та навігацію на початкову сторінку, використовуючи URL із налаштувань. Він забезпечує готовність середовища для виконання тестів. Крім того, клас реалізує інтерфейс IDisposable для забезпечення коректного завершення сесій драйвера та збереження логів після виконання тестів. Цей модуль забезпечує гнучкість у налаштуванні браузера й адаптованість до різних середовищ виконання, що робить платформу універсальним інструментом для автоматизованого тестування.

Модуль обробки вебелементів у платформі реалізований через набір класів, серед яких Element, Elements, ElementFinder, ElementsFinder, а також методи розширення WebElementExtension (Homes, 2024). Клас Element

ISSN 1728-3817

представляє окремий вебелемент і надає властивості й методи для взаємодії, включно з доступом до основного об'єкта типу IWebElement через webElementCore, отримання текстового значення елемента через Text, а також методи для кліку, очищення текстового поля та введення тексту. Додатково він підтримує функції автоматичного прокручування до елемента ScrollToElement та підсвічування елемента перед взаємодією PerformActionWithHighlighting, що підвищує стабільність і зручність тестування.

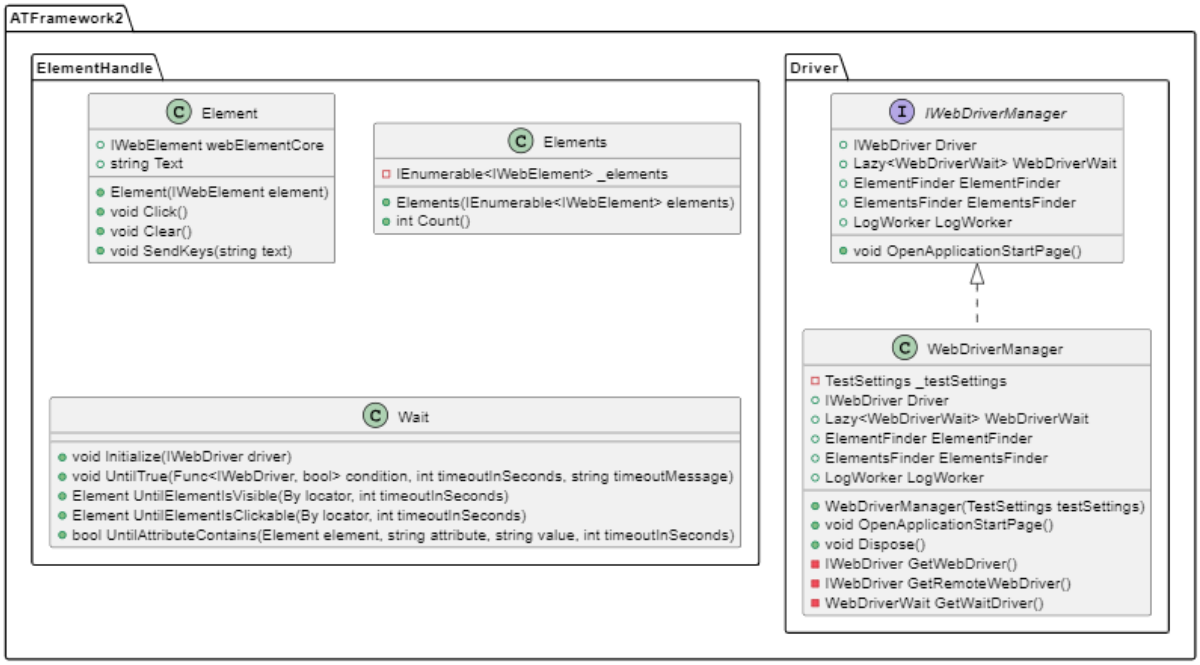

Рис. 1. Модулі ElementHandle та DriveManager

Клас Elements дає змогу працювати з колекціями вебелементів, забезпечуючи, наприклад, отримання кількості елементів у колекції через метод Count. Класи ElementFinder й ElementsFinder реалізують пошук одного або декількох елементів за допомогою CSS-селекторів, XPath або ID. Метод Css у ElementFinder шукає перший елемент за заданим CSS-селектором, повертаючи об'єкт Element, тоді як у ElementsFinder аналогічний метод повертає всі знайдені елементи у вигляді об'єкта Elements (Homes, 2024).

Методи розширення WebElementExtension додають функціональність до IWebElement, зокрема автоматичне прокручування до елемента, підсвічування під час взаємодії, а також переміщення курсора до центра елемента. Усі ці можливості доповнюють базовий функціонал Selenium WebDriver, забезпечуючи підвищену стабільність і зручність роботи з елементами. Завдяки модульній структурі цей компонент легко інтегрується з іншими частинами платформи, що робить його незамінним для автоматизації тестування.

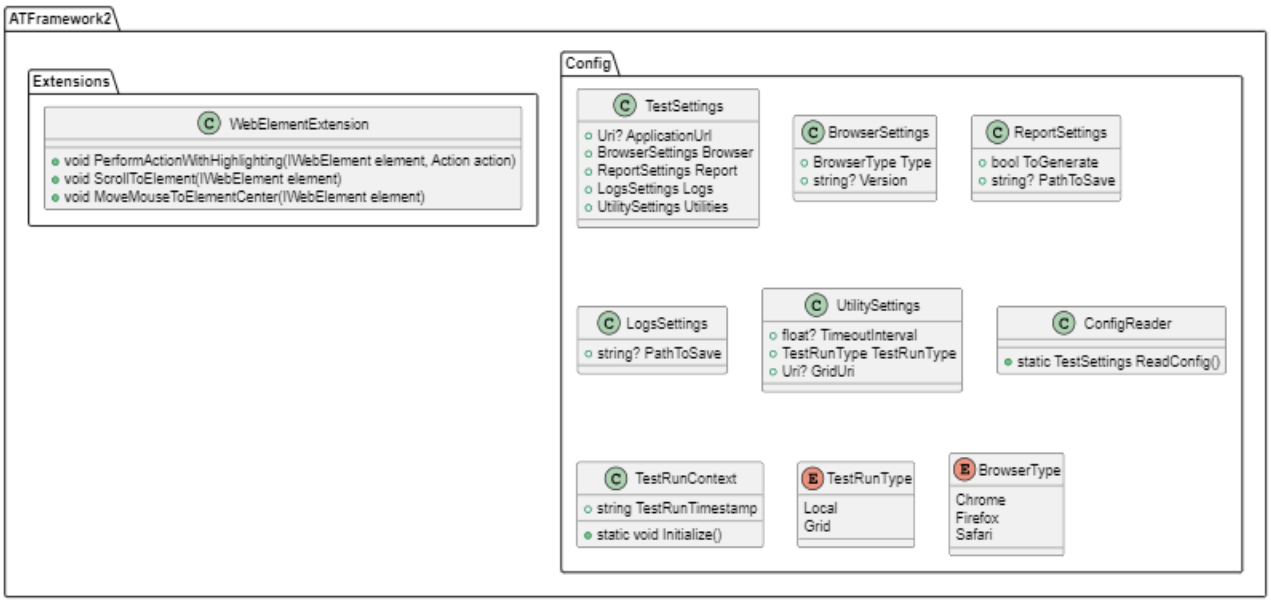

Рис. 2. Модулі Extensions і Configuration

Модуль Wait забезпечує стабільність тестів і коректну роботу з динамічними вебсторінками, реалізуючи механізм очікувань. Клас містить набір статичних методів для керування очікуваннями на основі WebDriverWait. Ініціалізація виконується через метод Initialize, що приймає екземпляр вебдрайвера, активує внутрішній об'єкт – driver і надає доступ до всіх функцій очікування.

Метод UntilTrue дає змогу чекати виконання заданої умови, що передається у вигляді функції condition, з можливістю встановлення тайм-ауту. У разі перевищення тайм-ауту генерується виняток із відповідним повідомленням. Методи UntilElementIsVisible й UntilElementIsClickable створені для роботи з вебелементами. Перший очікує появу елемента на сторінці та повертає його у вигляді об'єкта Element, другий перевіряє, коли елемент стане доступним для кліку.

Функціональність UntilAttributeContains дає змогу відстежувати зміни атрибутів елемента, очікуючи, поки атрибут отримає потрібне значення, що особливо корисно для контролю стану динамічних елементів.

Інтеграція цього механізму з іншими компонентами, зокрема з ElementFinder, сприяє ефективному поєднанню пошуку елементів з очікуваннями, що зменшує ймовірність помилок, спричинених асинхронною природою вебзастосунків. Завдяки цьому тестування стає надійнішим навіть у найскладніших сценаріях.

Модуль HtmlReportGenerator відповідає за створення детальних HTML-звітів про виконання тестів, надаючи інструменти для зручної йа інтерактивної візуалізації результатів тестування. Ініціалізацію виконують методом Instance, що створює або повертає вже наявний екземпляр цього класу. Реалізація за допомогою патерну Singleton гарантує, що для одного тестового запуску буде створено лише один екземпляр. Конфігураційні параметри зчитують із TestSettings, що дає змогу визначити необхідність генерації звітів і вказати шлях для їхнього збереження.

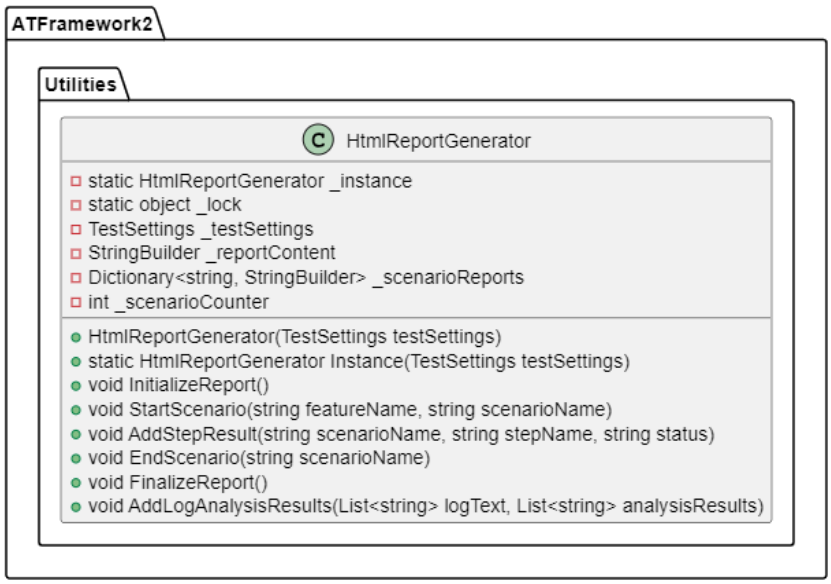

Рис. 3. Модуль HtmlReportGenerator

Формування базової структури HTML-документа виконують, використовуючи метод InitializeReport. Він додає заголовок, стилі CSS для оформлення таблиць, кнопок й інших елементів, а також JavaScript-код для інтерактивних функцій. Наприклад, collapsible-блоки дають можливість згортати й розгортати розділи для зручного перегляду деталей тестових сценаріїв.

Додавання нового сценарію до звіту здійснюється через метод StartScenario, що створює заголовок сценарію, ініціалізує таблицю для запису тестових кроків і присвоює унікальний ідентифікатор. Метод AddStepResult додає інформацію про виконані кроки, включно з їхніми назвами, статусом (Passed або Failed) і часовою міткою. Використовується кольорове виділення: зелений колір для успішних кроків і червоний – для невдалих. Після завершення сценарію метод EndScenario закриває відповідну таблицю.

Коли всі тестові сценарії виконані, метод FinalizeReport збирає всі дані в єдиний HTML-документ і зберігає його у вказаній директорії. Ім'я файлу містить часову мітку, що полегшує ідентифікацію конкретного тестового запуску.

Завдяки цій реалізації звіти є не лише інформативними, а й інтерактивними, надаючи детальну інформацію про кожен тест і спрощуючи аналіз результатів тестування.

Модулі CheckWorker і VerifyWorker забезпечують перевірку очікуваних результатів під час тестування, надаючи інструменти для контролю відповідності фактичних й очікуваних значень у тестових сценаріях. Його функціональність спрямована на підвищення гнучкості та стабільності тестів. Клас CheckWorker містить набір статичних методів для перевірки умов, таких як Equals, що дає змогу перевіряти рівність двох значень із підтримкою чутливості до регістру, зокрема: NotEquals, який перевіряє нерівність значень; Contains, що визначає наявність підрядка в рядку з опціональною підтримкою нечутливості до регістру; та Matches, який виконує перевірку за регулярними виразами. Специфічні методи, як DateTimeEquals і DateTimeNotEquals, забезпечують порівняння дат. Для динамічних середовищ метод ExecuteWithRetry допомагає повторювати перевірки до успіху або завершення часу очікування. Після завершення тестів функція FinalizeChecks збирає всі невдалі перевірки та надає виняток із деталізованим списком помилок.

Клас VerifyWorker надає розширені можливості, зокрема перевірку списків методом ListEqualsIgnoringOrder, який порівнює списки незалежно від порядку елементів. Метод Multiple дає змогу виконувати кілька перевірок одночасно, збираючи всі помилки в один результат. Логування результатів перевірок через LogWorker робить аналіз прозорим і зручним (Loubser, 2021).

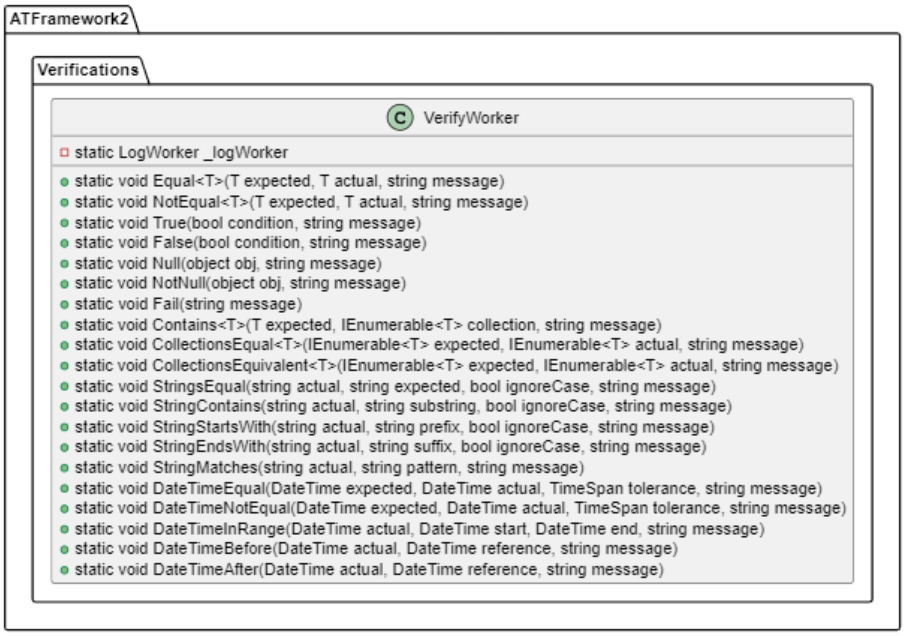

Рис. 4. Модуль VerifyWorker

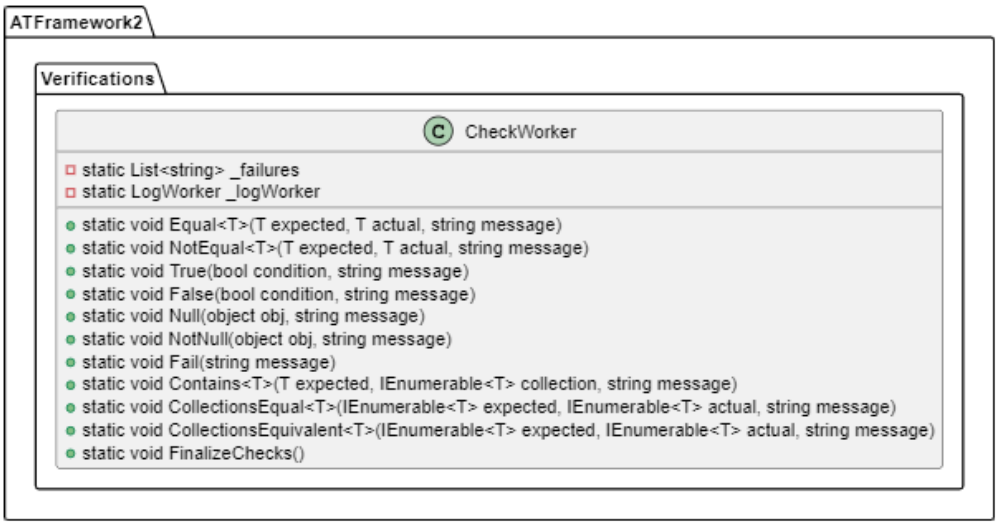

Рис. 5. Модуль CheckWorker

Обидва класи інтегровані з іншими компонентами платформи, такими як LogWorker для запису логів і Wait для повторних перевірок. Модуль верифікації є невід'ємною частиною платформи, забезпечуючи надійний контроль якості тестових сценаріїв і результатів, що сприяє стабільності й ефективності тестування.

Модуль ApiWorker забезпечує взаємодію з REST API, надаючи зручний і гнучкий інструмент для виконання HTTP-запитів й обробки відповідей у тестових сценаріях. Основою модуля є об'єкт HttpClient, що інкапсулює логіку виконання HTTP-запитів і допомагає гнучко налаштовувати поведінку, наприклад, через додавання middleware або налаштування тайм-аутів. Для створення екземпляра ApiWorker потрібно передати ініціалізований HttpClient, що дає можливість легко адаптувати модуль до специфічних вимог тестування (Mohan, 2022).

Підтримка основних CRUD-операцій у модулі реалізована через асинхронні методи, що забезпечують ефективну взаємодію з API. Метод GetAsync виконує HTTP-запит до вказаного endpoint і повертає результат у вигляді рядка, перевіряючи успішність операції за допомогою EnsureSuccessStatusCode. Для надсилання даних використовують PostAsync, який приймає JSON-дані у форматі StringContent із відповідним Content-Type. Оновлення ресурсів на сервері виконують методом PutAsync, що має аналогічний механізм роботи. Видалення даних реалізовано через DeleteAsync, який знадсилає запит на заданий endpoint.

Асинхронна реалізація всіх методів сприяє паралельному виконанню запитів у тестових сценаріях, підвищуючи продуктивність і гнучкість інтеграції. Завдяки реалізації інтерфейсу IDisposable клас ApiWorker дає змогу коректно завершувати роботу з HttpClient, вивільняючи ресурси після завершення тестування.

Зазначений модуль відіграє важливу роль у платформі, забезпечуючи можливість інтеграційного тестування, перевірки роботи API та легкого налаштування запитів відповідно до вимог тестування. Його взаємодія з іншими компонентами, такими як логування та верифікація, підвищує ефективність автоматизації та розширює функціональні можливості тестової платформи.

Модуль LocalStorageWorker надає інструменти для роботи з локальним сховищем браузера, забезпечуючи можливість маніпулювання даними, збереженими в локальному сховищі вебзастосунків. Його функціональність базується на виконанні JavaScript-кодів через IJavaScriptExecutor, що забезпечує прямий доступ до сховища.

Метод GetLocalStorage виконує JavaScript-код для зчитування всіх ключів і їхніх значень у Local Storage та повертає результат у вигляді словника Dictionary<string, string>. Це допомагає перевіряти дані, що зберігаються застосунком під час роботи. Метод AddToLocalStorage додає нові записи у вигляді пари ключ – значення, використовуючи JavaScript-функцію window.localStorage.setItem, що дає змогу підготувати стан браузера для тестів. Оновлення даних у сховищі виконується методом UpdateLocalStorageValue, що змінює значення для заданого ключа. Видалення записів реалізовано через метод DeleteFromLocalStorage, що використовує функцію removeItem, а повне очищення Local Storage виконується за допомогою методу ClearLocalStorage, який викликає window.localStorage.clear.

Цей модуль є важливим для тестування застосунків, що використовують Local Storage для збереження даних користувачів, налаштувань або сесій. Наприклад, перед виконанням тестів можна змінювати дані в сховищі, щоб створити необхідні початкові умови для тестових сценаріїв.

LocalStorageWorker легко інтегрується з іншими модулями платформи. Отримані дані можуть бути перевірені за допомогою модулів верифікації, таких як CheckWorker або VerifyWorker, або збережені в тестових звітах через HtmlReportGenerator. Це забезпечує платформу додатковою гнучкістю і допомагає тестувати складні сценарії роботи з вебзастосунками.

Модуль TestDataWorker сприяє створенню унікальних і динамічних тестових даних, що адаптуються до різних сценаріїв тестування, надаючи гнучкість і варіативність вхідних параметрів. Його використання зменшує залежність від статичних даних і підвищує гнучкість тестів.

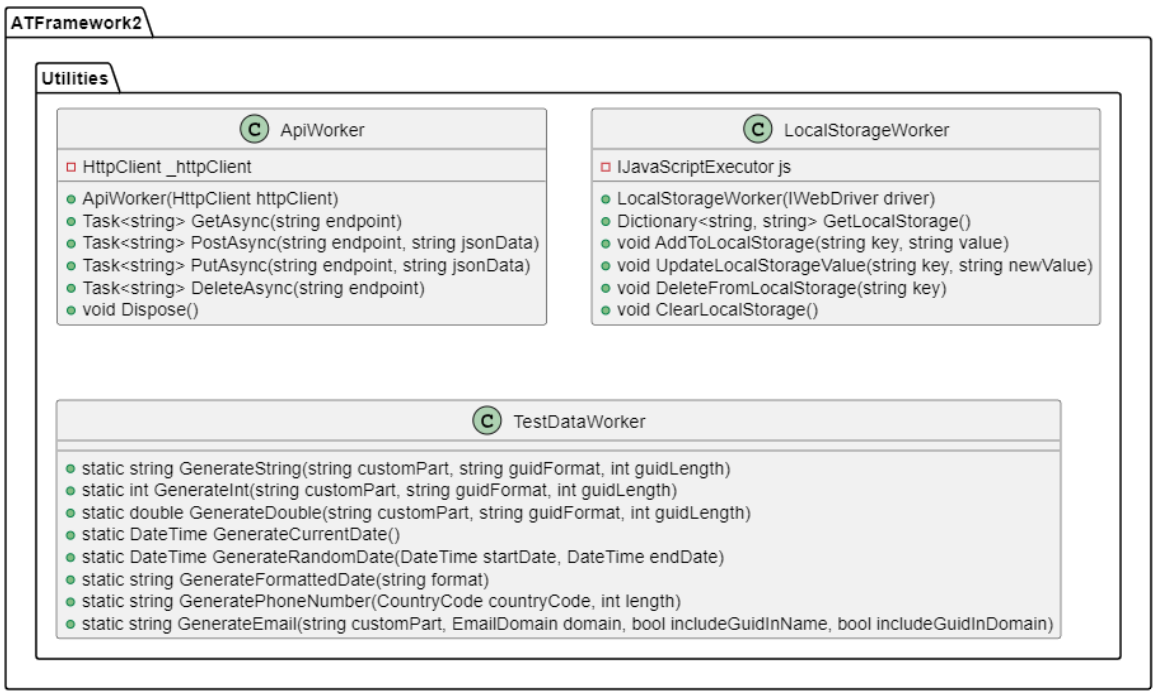

Рис. 6. Утилітарні компоненти платформи

Для генерації рядків і чисел метод GenerateString створює унікальний рядок, поєднуючи користувацьку частину та GUID із можливістю налаштування формату та довжини. Метод GenerateInt генерує унікальні цілі числа на основі хешування рядка з GUID, а GenerateDouble створює унікальні числа із плаваючою крапкою, комбінуючи рядок із хешованими значеннями. Для роботи з датами GenerateCurrentDate повертає поточну дату та час, GenerateFormattedDate дає змогу отримувати дати в заданому форматі, а GenerateRandomDate генерує випадкову дату в межах заданого інтервалу, перевіряючи коректність діапазону.

Генерація телефонних номерів здійснюється методом GeneratePhoneNumber, який створює номери із вказаним кодом країни, такими як +1 для США чи +380 для України, з можливістю налаштування довжини номера. Метод GenerateEmail продукує унікальні email-адреси, додаючи GUID до користувацької частини чи домену за необхідності, використовуючи вибраний домен, наприклад, example.com чи test.com.

Модуль зручний для автоматизації створення даних у різних тестових сценаріях, таких як реєстрація користувачів із використанням згенерованих email-адрес, тестування форм із випадковими датами або імітація унікальних телефонних номерів для перевірки локалізації. Згенеровані дані легко інтегруються з іншими модулями платформи, такими як модуль обробки елементів для заповнення форм чи ApiWorker для тестування API-запитів.

TestDataWorker є універсальним інструментом для динамічного налаштування тестових сценаріїв, що значно скорочує час їхньої підготовки й підвищує ефективність тестування.

Модуль LogWorker забезпечує відстеження процесу виконання тестів, збір діагностичної інформації та аналіз помилок, відіграючи вагому роль у платформі (Loubser, 2021). Реалізація цього компонента інтегрована у важливі модулі, зокрема CheckWorker, VerifyWorker, WebDriverManager і HtmlReportGenerator.

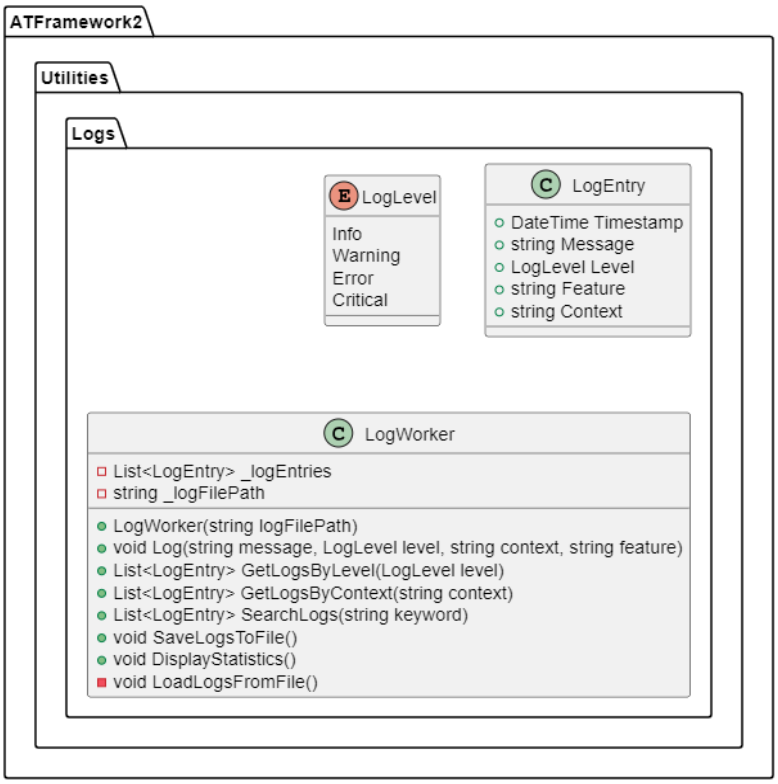

Рис. 7. Модуль логування

Основна функціональність LogWorker охоплює запис текстових повідомлень у лог-файли з додаванням часових міток, рівнів важливості (напр., Info, Error) та контексту виконання, що містить метод і клас, які викликали лог. Це допомагає створювати детальні записи подій для легкого ідентифікування джерела проблем. Повідомлення мають структурований вигляд, наприклад: [2024-12-14 12:34:56] [INFO] [WebDriverManager. OpenApplicationStartPage] Відкрито стартову сторінку: https://example.com, що забезпечує зручність читання й аналізу.

Для різних компонентів створюються окремі лог-файли, наприклад, VerifyWorkerLog.txt для перевірок або GlobalTestLogs_{timestamp}.txt для роботи з драйвером. Такий підхід дає змогу чітко відокремлювати інформацію про різні частини системи.

Логування тісно інтегроване з іншими модулями платформи. У CheckWorker і VerifyWorker записують як успішні перевірки, так і помилки із зазначенням очікуваних і фактичних значень. HtmlReportGenerator використовує логування для запису процесу створення звітів, фіксації помилок й інших ключових подій. У WebDriverManager логуються дії, пов'язані із запуском браузера, навігацією між сторінками та завершенням сесій драйвера. Усі логи зберігаються у директорії, зазначеній у LogsSettings.PathToSave, що спрощує доступ до них для подальшого аналізу.

Завдяки структурованим логам тестувальники можуть швидко ідентифікувати проблеми, оптимізувати сценарії тестування та надати докази результатів замовникам. Цей компонент підвищує надійність і підтримуваність платформи, роблячи її ефективною навіть у масштабних проєктах.

Інтеграція описаних модулів платформи в єдиний потік виконання є завершальним, але одним із найважливіших аспектів її роботи. Кожен модуль – від роботи з вебдрайвером, обробки елементів, верифікації, генерації звітів і до логування – органічно взаємодіє з іншими, забезпечуючи цілісність тестового процесу.

Потік починається з ініціалізації об'єкта TestSettings за допомогою модуля ConfigReader, який зчитує параметри з JSON-файлу для налаштування платформи відповідно до тестового запуску. Паралельно створюється контекст виконання TestRunContext із часовою міткою для подальшого відстеження дій. Після цього WebDriverManager створює вебдрайвер, вибираючи локальний чи віддалений режим роботи, а метод OpenApplicationStartPage відкриває стартову сторінку, визначену у TestSettings.ApplicationUrl. Усі дії логуються через LogWorker.

На наступному етапі модулі ElementFinder і ElementsFinder забезпечують пошук елементів для взаємодії, а Wait гарантує, що вони доступні. Через Element виконуються дії, як-от кліки чи введення тексту, із застосуванням методів розширення, що забезпечують стабільність і точність операцій. Результати дій перевіряються за допомогою модулів CheckWorker і VerifyWorker, які логують успішні перевірки та помилки.

Для інтеграційних тестів використовують ApiWorker, що надсилає HTTP-запити до бекенд-сервісів й аналізує відповіді. Усі важливі події, результати та статуси тестів фіксуються у лог-файлах через LogWorker, що забезпечує прозорість процесу. Завершальним етапом є генерація звітів через HtmlReportGenerator, який формує HTML-звіт із детальною інформацією про сценарії, кроки та результати виконання. Після завершення всіх тестів драйвер закривається, а зібрані дані зберігаються в заданих директоріях.

Інтеграція модулів платформи реалізована так, щоб забезпечити їхню взаємозалежність й узгоджену роботу, створюючи єдиний скоординований процес автоматизованого тестування. Наприклад, дані, знайдені ElementFinder, передаються в VerifyWorker для перевірки, результати верифікацій входять до звітів HtmlReportGenerator, а LogWorker записує всі ключові етапи роботи платформи.

Ця інтеграція забезпечує чіткий потік виконання, автоматизуючи всі аспекти тестування. Завдяки цьому платформа залишається гнучкою, масштабованою та простою у використанні навіть для складних проєктів, відповідаючи вимогам різних тестових сценаріїв.

2. Функціональні особливості платформи

Запропонована платформа для автоматизації тестування вебзастосунків є сучасним рішенням, що поєднує ефективність, стабільність і масштабованість у єдиному підході до тестування. Вона створена з урахуванням проблем, з якими стикаються тестувальники під час роботи з традиційними інструментами, та пропонує такі можливості для більш гнучкої та точної автоматизації:

1. Гнучка модульна архітектура є важливою особливістю платформи, оскільки вона дає змогу інтегрувати нові функціональні можливості без необхідності зміни основного коду. На відміну від багатьох рішень, що мають жорстку архітектурну структуру та вимагають значних змін при адаптації до нового середовища, ця платформа забезпечує легкість налаштувань і масштабування завдяки чітко визначеним компонентам. Це особливо важливо для великих проєктів, де необхідно швидко адаптувати тестове середовище до змін у вебзастосунку.

2. Підтримка різних типів тестування є однією із фундаментальних переваг платформи. Платформа дає можливість виконувати не лише тестування користувацького інтерфейсу (UI-тести), але й API-тестування, що є критичним для багаторівневих вебзастосунків. Завдяки цьому вона допомагає тестувальникам перевіряти як безпосередню взаємодію користувача з вебінтерфейсом, так і правильність роботи викликів API, що забезпечує комплексний підхід до перевірки програмного забезпечення.

3. Динамічне налаштування параметрів тестування через JSON-файли дає змогу легко змінювати середовище виконання тестів без необхідності вносити зміни до коду, що забезпечує гнучкість і зручність у використанні платформи. Це особливо корисно під час виконання тестування в різних середовищах, таких як локальне, тестове, проміжне або клієнтське, на відміну від традиційних платформ, що вимагають ручного внесення змін у код або використовують складні системи конфігурацій, що значно уповільнює процес налаштування тестів.

4. Наявність різних режимів запуску тестів, включно з локальним виконанням і запуском через Selenium Grid, сприяє ефективному використанню платформи в розподілених системах тестування. Це особливо важливо для великих команд або організацій із великою кількістю тестових сценаріїв, оскільки забезпечує можливість паралельного виконання тестів на різних пристроях і браузерах.

5. Механізм управління браузерними драйверами є перевагою платформи, оскільки використання патерну Singleton дає змогу уникнути зайвого створення браузерних сесій, що значно підвищує стабільність тестів та оптимізує використання ресурсів. У конкурентних платформах часто зустрічається проблема надмірного запуску браузерів, що призводить до перенавантаження тестового середовища та нестабільної роботи тестів. Завдяки правильному керуванню ресурсами й інтелектуальному очікуванню (Lazy WebDriverWait) платформа не перенавантажує систему і працює значно швидше.

6. Модуль обробки вебелементів забезпечує можливість взаємодії з елементами сторінок, спрощуючи виконання автоматизованих тестів. Убудовані механізми автоматичного прокручування до елементів та їхнє підсвічування під час виконання дій зменшують кількість тестових помилок, пов'язаних із невидимими або недоступними елементами. Це особливо корисно для складних динамічних вебзастосунків, у яких стандартні інструменти взаємодії з елементами можуть працювати нестабільно. Розроблювана платформа забезпечує глибший контроль над роботою з елементами, що підвищує точність і надійність тестування, на відміну від багатьох інших платформ, які обмежуються лише базовою взаємодією з інтерфейсом.

7. Наявна система очікувань не використовує жорстко задані таймінги, а дає можливість встановлювати динамічні умови очікування. Це особливо важливо для вебзастосунків, що активно використовують асинхронні запити та динамічний рендеринг контенту. На відміну від інших платформ, що часто вимагають ручного налаштування очікувань, цей підхід спрощує підтримку тестів і підвищує їхню стабільність.

8. Автоматично згенеровані інтерактивні HTML-звіти містять детальну інформацію про кожен тестовий сценарій, включно зі статусами виконання, скріншотами помилок та інтерактивними елементами для навігації. Це суттєво спрощує аналіз тестових запусків, у той час як багато інших рішень обмежуються лише текстовими або статичними звітами.

9. API-тестування є важливою особливістю платформи, оскільки забезпечує можливість автоматизованої перевірки роботи вебсервісів без необхідності ручного втручання. На відміну від платформ, які вимагають інтеграції з окремими інструментами для тестування API, запропонована система має вбудований API-модуль, що надає можливість проводити повноцінне тестування REST-інтерфейсів без додаткових налаштувань. Завдяки реалізації підтримки асинхронного виконання запитів, API-тести виконуються значно швидше, ніж під час використання традиційних рішень.

10. Можливості верифікації даних у платформі є розширеними, що сприяє здійсненню детальної перевірки коректності отриманих результатів у тестах. Убудовані модулі перевірки результатів підтримують автоматичний повтор перевірок у випадку невідповідностей, що забезпечує гнучкість під час тестування динамічних інтерфейсів. Це допомагає зменшити кількість хибно виявлених помилок у тестах і забезпечує точнішу верифікацію функціональності.

11. Підтримка роботи з локальним сховищем браузера дає змогу ефективно тестувати сценарії, пов'язані зі збереженням і використанням даних на рівні браузера. Це усуває потребу у використанні зовнішніх скриптів або ручного маніпулювання даними у сховищі.

Завдяки модульності архітектури, інтеграція штучного інтелекту не потребуватиме суттєвих змін у кодовій базі та дає змогу поступово розширювати інтелектуальні можливості платформи, додаючи нові алгоритми для поліпшення автоматизації тестування. Подальший розвиток платформи передбачає впровадження штучного інтелекту для аналізу тестових логів. Використання машинного навчання надасть можливість автоматично класифікувати та пріоритизувати помилки, аналізувати закономірності у збоях і прогнозувати потенційні проблеми. Це принесе додаткові можливості у сферу автоматизації тестування, оскільки допоможе зменшити навантаження на тестувальників і підвищити ефективність усунення дефектів.

Запропонована платформа забезпечує стабільність, продуктивність і зручність у використанні. Завдяки своїй гнучкості, інтегрованості та можливостям розширення, очікується, що вона може стати ефективним інструментом як для малих команд, так і для великих проєктів. Ураховуючи описану функціональність, ця система має значний потенціал для подальшого розвитку й упровадження нових інновацій у сфері автоматизації тестування.

Дискусія і висновки

Запропонована платформа для автоматизованого тестування вебзастосунків є комплексним рішенням, що поєднує можливості UI- та API-тестування, підтримує кросбраузерну взаємодію та допомагає масштабовано проводити перевірки функціональності вебзастосунків. Основною її особливістю є:

- модульна архітектура – легка адаптація, інтеграція нових компонентів і зміна параметрів тестування без редагування коду;
- комплексний підхід – поєднання UI- та API-тестування в єдиному середовищі;
- гнучкість – проста інтеграція нових функцій без змін у базовій структурі;
- централізоване управління конфігураціями – можливість швидкої зміни параметрів тестування без втручання в код;
- автоматичні HTML-звіти – детальна аналітика, скріншоти та логування для прозорості тестування.

Зазначена платформа також вирізняється високою стабільністю. Завдяки продуманій системі очікувань, розширеним механізмам роботи з елементами та налаштовуваним параметрам очікування, вона надає можливість ефективно працювати навіть із динамічними вебзастосунками. Це робить її придатною для тестування складних вебзастосунків, що активно використовують асинхронні запити і динамічний контент.

Масштабованість платформи забезпечується підтримкою локального та віддаленого запуску тестів, що дає змогу використовувати її як у невеликих командах, так і в масштабних корпоративних проєктах. Вона легко інтегрується з іншими інструментами, що розширює можливості її використання.

Завдяки зручному налаштуванню параметрів тестування та простоті у використанні платформа підходить для тестувальників із різним рівнем підготовки. Вона дає змогу швидко розпочати автоматизацію тестування без необхідності глибокого занурення в технічні деталі, що значно скорочує час упровадження автоматизованих перевірок у процес розроблення.

Представлена автором платформа є сучасним, функціональним і перспективним інструментом для автоматизації тестування, який може бути ефективно використаний у широкому спектрі проєктів. Подальший розвиток, зокрема інтеграція методів штучного інтелекту для аналізу логів та автоматичної класифікації помилок, допоможе значно підвищити продуктивність процесу тестування, прискорити виявлення дефектів і поліпшити якість програмного забезпечення. Завдяки модульності архітектури, така інтеграція не вимагатиме суттєвих змін у кодовій базі, що сприятиме поступовому розширенню можливостей платформи відповідно до нових викликів у сфері тестування.

Джерела фінансування. Дослідження частково профінансоване Київським національний університетом імені Тараса Шевченка.

Список використаних джерел

Forgacs, I., & Kovacs, A. (2024). *Modern Software Testing Techniques: A Practical Guide for Developers and Testers* (1st ed.). Apress.
Homes, B. (2024). *Fundamentals of Software Testing* (2nd ed.). Revised and Updated. Wiley-ISTE.
Mohan, G. (2022). *Full Stack Testing: A Practical Guide for Delivering High Quality Software*. O'Reilly Media.
Loubser, N. (2021). *Software Engineering for Absolute Beginners: Your Guide to Creating Software Products* (1st ed.). Apress.

References

Forgacs, I., & Kovacs, A. (2024). *Modern Software Testing Techniques: A Practical Guide for Developers and Testers* (1st ed.). Apress.
Homes, B. (2024). *Fundamentals of Software Testing* (2nd ed.). Revised and Updated. Wiley-ISTE.
Mohan, G. (2022). *Full Stack Testing: A Practical Guide for Delivering High Quality Software*. O'Reilly Media.
Loubser, N. (2021). *Software Engineering for Absolute Beginners: Your Guide to Creating Software Products* (1st ed.). Apress.

Отримано редакцією журналу / Received: 23.03.25
Прорецензовано / Revised: 15.08.25
Схвалено до друку / Accepted: 10.10.25

Danyil LIPSKYI, PhD Student
ORCID ID: 0009-0000-4068-9453
e-mail: lipsky@knu.ua
Taras Shevchenko National University of Kyiv, Kyiv, Ukraine

ARCHITECTURE AND FUNCTIONAL CAPABILITIES OF A PLATFORM FOR AUTOMATED WEB APPLICATION TESTING

The article presents a platform for automated web application testing based on an object-oriented approach using C#. The platform is designed for scalable and flexible testing of web applications, supporting both UI and API testing capabilities. The main system components are described and analyzed in terms of their functionality and role in automation. The structure and interaction of components are examined in detail, highlighting efficient browser driver management, test environment configuration, and the collection and processing of test data. A distinctive feature is the use of the singleton pattern for browser driver management and support for cross-browser testing, allowing validation of web applications across different platforms. The platform also supports dynamic configuration and extensibility through configuration files and centralized parameter management. The system provides convenient mechanisms for locating web elements and their collections and extends functionality for API integration testing and working with the browser's local storage. The platform is geared toward automating both scenario-based and utility testing, contributing to an effective approach to verifying the functionality and stability of web applications.

K e y w o r d s : ***automated testing, web applications, C#, cross-browser testing, environment configuration, UI testing, API testing.***

Автор заявляє про відсутність конфлікту інтересів. Спонсори не брали участі в розробленні дослідження; у зборі, аналізі чи інтерпретації даних; у написанні рукопису; в рішенні про публікацію результатів.

The author declares no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses or interpretation of data; in the writing of the manuscript; in the decision to publish the results.