Міністерство освіти і науки України
Кам’янець-Подільський національний університет імені Івана Огієнка
Фізико-математичний факультет
Кафедра комп'ютерних наук

Кваліфікаційна робота магістра
з теми: «**Модель масштабованої архітектури фреймворку автоматизації тестування адаптивних вебзастосунків**»

Виконав: здобувач вищої освіти
групи KN1-M24
спеціальності 122 Комп’ютерні науки
Косінов Михайло Сергійович

Керівник:
Мястковська Марина Олександрівна,
кандидат педагогічних наук, старший
викладач кафедри комп’ютерних наук

Науковий консультант:
Іванюк Віталій Анатолійович, доктор
технічних наук, доцент, завідувач
кафедри комп’ютерних наук

Рецензент:
Кушнір Оксана Климівна, кандидат
економічних наук, доцент, доцент
кафедри економіки підприємства

Кам’янець-Подільський – 2025 р.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

API – application programming interface

BDD – behavior-driven development

KDT – keyword driven testing

POM – page object model

АСК – автоматизована система керування

ЗВО – заклад вищої освіти

ПЗ – програмне забезпечення

АНОТАЦІЯ

Косінов М.С., здобувач вищої освіти. Модель масштабованої архітектури фреймворку автоматизації тестування адаптивних вебзастосунків. – Кваліфікаційна робота на правах рукопису.

Науковий керівник: Мястковська М.О., кандидат педагогічних наук, старший викладач кафедри комп'ютерних наук.

Науковий консультант: Іванюк В.А., доктор технічних наук, доцент, завідувач кафедри комп'ютерних наук.

Кваліфікаційна (магістерська) робота на здобуття ступеня магістра за спеціальністю 122 Комп'ютерні науки. – Кам'янець-Подільський Національний університет імені Івана Огієнка, Кафедра комп'ютерних наук, Кам'янець-Подільський, 2025.

Метою кваліфікаційної роботи є створення моделі масштабованої архітектури фреймворку автоматизації тестування адаптивних вебзастосунків.

Для досягнення мети у роботі розглянуто сучасні тенденції та підходи до автоматизації тестування, проведено аналіз і порівняння існуючих інструментів тестування вебзастосунків. У роботі обґрунтовується використання Playwright у вигляді основного інструменту над яким було побудовано модель масштабованої архітектури фреймворку автоматизації тестування адаптивних вебзастосунків. Продемонстровано використання і реалізацію сучасних підходів автоматизації тестування під час виконання практичної частини роботи. Тестування моделі за допомогою існуючого адаптивного вебзастосунку демонструє ефективність моделі та свідчить про широкі перспективи та вигоду від впровадження у комерційні проєкти.

Ключові слова: автоматизація тестування, адаптивні вебзастосунки, модель архітектури, повернення інвестицій, проєктування, конфігурування.

ABSTRACT

Kosinov M.S., Higher Education Applicant. A Model of Scalable Architecture for the Test Automation Framework of Adaptive Web Applications. –

Qualification Thesis as a Manuscript.

Scientific Advisor: Miastkovska M.O., Candidate of Pedagogical Sciences, Senior Lecturer of the Computer Science Department.

Scientific Consultant: Ivaniuk V.A., Doctor of Technical Sciences, Associate Professor, Head of the Department of Computer Science.

Qualification (Master's) Thesis for the Master's Degree in Specialty 122 Computer Science. – Kamianets-Podilskyi Ivan Ohiienko National University, Computer Science Department, Kamianets-Podilskyi, 2025.

The goal of the qualification thesis is to create a model of scalable architecture for the test automation framework of adaptive web applications.

To achieve this goal, the work examines modern trends and approaches to test automation, conducts an analysis and comparison of existing web application testing tools. The work substantiates the use of Playwright as the main tool upon which the model of scalable architecture for the test automation framework of adaptive web applications was built. The use and implementation of modern test automation approaches were demonstrated during the practical part of the work. Testing the model using an existing adaptive web application demonstrates the effectiveness of the model and indicates wide prospects and benefits from its implementation in commercial projects.

Keywords: test automation, adaptive web applications, architecture model, return on investment, design, configuration.

ВСТУП

Актуальність. Результат інформаційно-технологічної революції вносить свої корективи у діяльність організацій у всіх сферах економіки держав та побут кожної родини. Автоматизоване виробництво, торговельні та логістичні операції, маркетингові кампанії, освітні траєкторії, моніторинг і корекція показників здоров'я людини та безліч інших процесів відбуваються за допомогою цифрових продуктів та автоматизованих систем. З розвитком складності комп'ютерних програм виникає потреба належного їх тестування, оскільки некоректна робота програм може призвести до значних фінансових збитків та, у гіршому випадку, людських смертей. Значні витрати коштів та часу на розробку, підтримку та тестування програм потребують оптимізації з метою створення послуг доступних для багатьох користувачів. Для досягнення цієї мети тестування складних мережевих запитів, значної кількості компонентів інтерфейсу та іншої функціональності можуть бути автоматизовані.

Сьогодні адаптивні вебзастосунки є найпопулярнішим типом програмного забезпечення та мають широкий перелік інструментів для автоматизації тестування, таких як Selenium, Cypress, Playwright, Cucumber. Дані інструменти надають певний перелік бібліотек та методів, які дозволяють взаємодіяти з вебзастосунками, але немає комплексного рішення, яке забезпечить початкові налаштування і необхідний функціонал для отримання результатів тестування.

Основними перевагами автоматизованого тестування є мінімізація людських помилок, зменшення часових витрат на виконання регулярно повторюваних дій та тестів. Зважаючи на важливість процесів тестування, необхідно здійснювати постійні контроль та перевірку за результатами автоматизованого тестування, які полягають у створенні кодової бази для взаємодії з застосунком, який тестується, формуванні електронних звітів, лог-файлів (для перевірки роботи окремих компонентів і функцій фреймворку автоматизації) та, за вимогою, інтеграції з інструментами CI/CD та іншими.

Для забезпечення високої якості автоматизованого тестування адаптивних вебзастосунків необхідне комплексне рішення, що поєднує можливості сучасних фреймворків зі сторонніми бібліотеками для формування звітів про виконання тестів та інтеграції з CI/CD сервісами.

Мета дослідження – розробка моделі масштабованої архітектури фреймворку автоматизації тестування адаптивних вебзастосунків.

Для досягнення мети було визначено наступні **завдання**:

1. проаналізувати сучасні проблеми та особливості автоматизованого тестування вебзастосунків;
2. дослідити існуючі фреймворки, призначені для тестування адаптивних вебзастосунків;
3. розробити модель масштабованої архітектури фреймворку автоматизації тестування адаптивних вебзастосунків;
4. реалізувати та дослідити ефективність розробленої моделі.

Об’єкт дослідження – процес автоматизованого тестування вебзастосунків.

Предмет дослідження – інструменти автоматизованого тестування адаптивних вебзастосунків.

Методи дослідження:

1. Аналіз і синтез: для виявлення закономірностей у сучасних підходах до забезпечення якості та визначення вимог до компонентів моделі фреймворку, що проєктується.
2. Порівняльний аналіз: для зіставлення існуючих інструментів автоматизації (Selenium, Cypress, Playwright) з метою обрання найбільш доцільного інструменту.
3. Моделювання: для розробки структури та демонстрації взаємозв’язків між компонентами масштабованої архітектури фреймворку (забезпечення модульності між різними рівнями моделі).
4. Експериментальний метод: для перевірки працездатності розробленого фреймворку на прикладі реального вебзастосунку, оцінки

масштабованості моделі та стабільності тестів.

5. Спостереження: для аналізу поведінки фреймворку та підтримки адаптивності під час виконання тестів для вебзастосунків та моніторингу результатів у згенерованих звітах (HTML/Allure).

Наукова новизна. Розроблено нову модель масштабованої архітектури фреймворку автоматизації тестування адаптивних вебзастосунків, яка, на відміну від існуючих рішень, базується на чотирирівневій структурі з виокремленням шару тестових даних та наскрізною взаємодією бізнес-логіки з усіма компонентами системи, що забезпечує гнучку адаптацію тестів під різні платформи (mobile, tablet, desktop) без дублювання коду.

Практичне значення одержаних результатів: створена модель масштабованої архітектури фреймворку автоматизації тестування адаптивних вебзастосунків може бути використана для комерційних потреб з метою зменшення вартості впровадження автоматизованого тестування. Також, модель може бути використана в освітньому процесі ЗВО з метою навчання та фахової підготовки фахівців технічних спеціальностей.

Результати роботи впроваджені в роботу кафедри комп’ютерних наук Кам’янець-Подільського національного університету імені Івана Огієнка.

Апробація одержаних результатів:

1. Виступ на XVIII Міжнародній науково-практичній онлайн інтернет-конференції «Проблеми та інновації в математичній, цифровій, природничій і професійній освіті» (Центральноукраїнський державний університет імені Володимира Винниченка, 20-27 листопада 2024 р.) [1].

2. Виступ на Міжнародній науково-практичній конференції «Цифрова трансформація освіти: теоретико-методичні засади», присвяченій 70-річчю від дня народження доктора педагогічних наук, професора, директора Навчально-наукового інституту перепідготовки та підвищення кваліфікації Володимира Сергієнка (Київ, 28 жовтня 2024 р.).

3. Виступ на III Міжнародній науково-практичній інтернет-конференції «Актуальні аспекти розвитку STEAM-освіти в умовах євроінтеграції»

присвяченій 64-й річниці Донецького державного університету внутрішніх справ. 24 квітня 2025 року. м. Кропивницький [2].

4. XIX Міжнародна науково-практична інтернет конференція «Проблеми та інновації в математичній, цифровій, природничій і професійній освіті» Центральноукраїнського державного університету імені Володимира (20 травня – 29 травня 2025 р.) [3].

5. Участь у науковій конференції студентів і магістрантів за підсумками НДР у 2024-2025 н.р. (9-10 квітня 2025 р.) [4].

Результати роботи опубліковані у 3 тезах [1, 2, 3] та 3 статтях [4, 5, 6].

Структура роботи: вступ, три розділи, висновки, список використаних джерел (тридцять чотири) та додатки. Також у роботі наведено 18 ілюстрацій та продемонстровано лістинги програмного коду.

РОЗДІЛ 1. ТЕНДЕНЦІЇ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ АДАПТИВНИХ ВЕБЗАСТОСУНКІВ

Розвиток галузі інформаційних технологій здійснив значний вплив на життя людини: електронний документообіг, електронні державні реєстри, соціальні мережі, обладнання для моніторингу і здійснення контролю за різними явищами. Все це сприяє розширенню доступних платформ: мобільні, десктопні, веб, хмарні та автоматизовані пристрої та сервіси, вбудоване ПЗ (embedded) та інші [1]. Для середньостатистичного користувача важливо мати безперебійний доступ до необхідної інформації, саме тому використання мобільних пристроїв здобуло значної популярності. Проте, для здійснення адміністрування та опрацювання даних зручно використовувати стаціонарні системи (ноутбук, ПК), які дозволяють за потребою швидко перемикатися між різними інструментами. З метою зменшення витрат на розробку і впровадження державні та бізнес-установи схиляються до використання адаптивних вебзастосунків, які потребують належного тестування для утримання користувачів і задоволення їх потреб.

1.1. Сучасні тенденції тестування веб-застосунків

У сучасних умовах розробки ПЗ, де вимоги до якості, швидкості і відмовостійкості постійно зростають, класична модель тестування не в змозі задовольнити визначені потреби у повному обсязі. Мануальне тестування залишається невід’ємним етапом визначення якості програмного продукту, проте ризики пов’язані зі схильністю до людських помилок, трудомісткістю, часовими і вартісними витратами не забезпечують належного покриття функціоналу тестами. У відповідь на ці виклики, логічним кроком стала автоматизація тестування, метою якої є зниження витрат на реалізацію програмного продукту, мінімізація людських помилок і зниження об’ємів рутинних завдань.

Відповідно, такий підхід вніс свої зміни у традиційні підходи

тестування, створивши нові потреби і завдання перед інженерами контролю якості програмних продуктів [1]. Серед таких завдань відзначають:

- визначення пріоритетів тестування;
- узгодження цілей тестування та оцінка ризиків;
- вибір відповідних інструментів та фреймворків для тестування;
- визначення та аналіз витрат на автоматизацію тестування;
- оновлення і підтримка тестових даних та середовищ;
- забезпечення стабільного виконання тестів;
- інформативне звітування для аналізу результатів тестування.

У сучасних вебсистемах важливою вимогою є забезпечення коректної роботи системи на різних конфігураціях, які включають в себе клієнтські пристрої, браузери, сервери, належні до різних країн з різними законодавчими вимогами, тощо. Таким чином, кількість одних і тих самих тестів, які повинні бути виконані значно зростає.

Виконання і управління великою кількістю тестових сценаріїв з більшою ймовірністю призведе до більшої неефективності. Саме тому, у сучасній автоматизації тестування значну увагу приділяють стратегічному плануванню, пріоритизації тестів, які необхідно автоматизувати, та оптимізації процесів. Одним з ключових факторів, який впливає на визначення необхідності впровадження автоматизації у проєкт є обрахування показника рентабельності інвестицій.

ROI (англ. “return on investment” – повернення інвестицій) – показник, який оцінює фінансову вигоду та переваги ефективності, отримані від впровадження автоматизованого тестування програмного забезпечення, у порівнянні з загальними витратами [7]. Для досягнення кращих показників повернення інвестицій необхідним етапом є стратегічне планування процесів автоматизованого тестування, що вимагає глибокого розуміння доменної області, функціональної специфіки проєкту та поточний стан його реалізації. Врахування вартісних та часових витрат на створення інфраструктури та написання тестових скриптів є одним з ключових факторів. Наприклад,

складні і часто змінювані сценарії потребуватимуть більше часу на їх оновлення, стабілізацію та підтримку, що негативно вплине на загальну вартість автоматизованого тестування.

Коефіцієнт фінансової вигоди від впровадження автоматизованого тестування може обчислюватися за формулою:

$$ROI(\%) = \frac{(\text{Вигоди від автоматизації} - \text{Витрати на автоматизацію})}{\text{Витрати на автоматизацію}} \times 100$$

Таким чином, впровадження автоматизації у проєкт є доцільним, коли витрати на автоматизацію дозволяють знизити загальні витрати проєкту, у порівнянні з тим, що увесь процес тестування виконувався б мануально. Відповідну залежність продемонстровано на рис 1.1

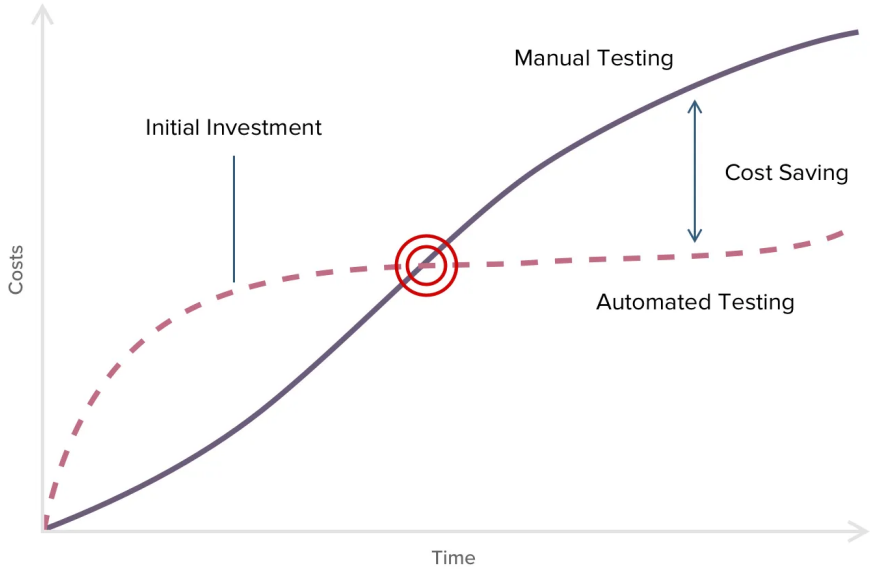


Рис. 1.1. Очікувані фінансові вигоди від впровадження автоматизації.

Джерело: Auto testing ROI. Режим доступу:

https://www.cleveroad.com/blog/test-automation-roi/

Архітектура сучасних вебзастосунків все частіше реалізується за допомогою SPA (англ. “single page application” – односторінкові застосунки) з використанням React, Angular, Vue.

З метою визначення ефективного і раціонального шляху, необхідного для реалізації масштабованої архітектури фреймворку автоматизації тестування адаптивних вебзастосунків, який матиме практичну цінність,

необхідним етапом є здійснення детального аналізу існуючих особливостей і викликів, що виникають в процесі розробки і тестування сучасних вебзастосунків.

Варіативність, насиченість і складність користувальницького інтерфейсу створюють більшу кількість варіантів тестових сценаріїв та, як наслідок, ускладнюють процес тестування. Другою, складністю є те, що сучасні додатки повинні працювати на різноманітних пристроях та операційних системах. Це призводить до того, що процес тестування стає об'ємним та довготривалим [8]. Враховуючи перелічені особливості веб-систем постає необхідність перегляду існуючих підходів до тестування, щоб задовольнити принципи тестування, серед яких: тестування залежить від контексту; тестування показує наявність дефектів; раннє тестування; вичерпне тестування неможливе; скупчення дефектів; парадокс пестицидів; омана відсутності помилок [9].

Спираючись на матеріали розглянутого дослідження, про особливості тестування сучасних вебсистем та принципи тестування, очевидно, що складність процесів розробки вебсистем ставить тестувальників в умови, коли їм необхідно змінювати стратегію та підходи до здійснення тестування, з метою дотримання принципів тестування та забезпечення належної якості програмного продукту. Важливим кроком для побудови ефективної автоматизації є обрання найбільш доцільного підходу до автоматизації тестування та інструментів. Огляд існуючих рішень показує, що є значна розбіжність між комерційними рішеннями та академічними дослідженнями в галузі автоматизованого тестування, оскільки значна кількість академічних досліджень не була протестована у реальних практичних умовах реалізації проєктів, що становить під загрозу практичну цінність розглянутих підходів [8, 10]. Окрім класичних і поширених інструментів розглянуто академічні інструменти "Robula+", "Apogen", "Vista" [11, 12, 13, 14]. Було розглянуто основні підходи, до яких належать "Page object model", "Behavior driven development", "Keyword driven testing" [16, 17, 18, 19]. Особливості існуючих

підходів та академічних інструментів автоматизованого тестування, які включають в себе AI-інструменти, наведено у додатку А.

1.2. Аналіз проблем і викликів сучасних підходів

Проведений огляд таких патернів як “Page object model”, “Behavior driven development”, “Keyword driven development” надає певний перелік відомостей для проведення аналізу, який дозволить визначити перспективні напрями для вдосконалення та покращення процесів тестування. Спрямованість на вирішення проблем, які є властивими для певних типів проєктів та залежність від конкретних платформ створюють обмеження використання наявних інструментів у широкому просторі, а особливості реалізації – нові потреби у підготовці та перекваліфікації фахівців.

Використання комбінованих підходів дозволить поєднати переваги та знизити рівень значущості недоліків кожного з підходів. Орієнтація на формування і вдосконалення бізнес процесів у автоматизації тестування потребує досягнення наступних важливих аспектів на проведення автоматизації тестування: зниження вартості автоматизації, зниження часових витрат на проведення тестування, поліпшення комунікації між технічним і нетехнічними спеціалістами різного рівня професійних умінь, створення і реалізація підходів для зменшення кількості залежностей у процесах автоматизації, оптимізація процесів впровадження нового і підтримки існуючого функціоналу систем.

Підхід POM є найбільш поширеним у тестуванні вебзастосунків, незважаючи на наявні у ньому недоліки, серед яких: складність обробки динамічних елементів, ускладнення підтримки фреймворку автоматизації тестування відповідно до етапів розвитку проєкту, початкове налаштування фреймворку [19].

Додатковим викликом автоматизації тестування веб-застосунків є те, що більшість з них забезпечують адаптивність роботи на різних типах пристроїв, що пропорційно збільшує часові витрати на проведення тестування та

збільшує ймовірність допущення людських помилок.

Зважаючи на наявні проблеми і виклики в автоматизації тестування сучасних веб-систем пошук нових та/або вдосконалення існуючих підходів є необхідним для забезпечення належної якості програмного продукту.

1.3. Необхідність створення моделі масштабованої архітектури

Опис функціональних можливостей сучасних фреймворків автоматизації тестування для вебзастосунків має широкий перелік функцій та API, які постачаються тестувальникам [5]. Проте, характерною рисою цих інструментів є те, що вони містять в собі базові функції для взаємодії з різними елементами веб-сторінок, з базовою файловою структурою проєкту, яку розробники адаптують під власні потреби. Це ставить розробників у становище, коли необхідно здійснювати аналіз і пошук додаткових бібліотек, що потребує додаткових вартісних і часових витрат.

Структуризація і розподіл кодової бази, як правило ділиться на дві основні компоненти: основна кодова база та автоматизовані тестові скрипти. Реалізація кодової бази потребує детального планування і врахування можливостей для масштабування, з метою уникнення витрат на проведення раннього рефакторингу.

Клієнт-серверна взаємодія – основа для тестування веб-застосунків. Перевірка функціональності бекенду, інтеграції між різними компонентами потребують взаємодії з різними сутностями, обробки даних різного формату. Для реалізації належного автоматизованого тестування необхідним є механізм надсилання та обробки запитів до серверу.

Реалізація інформативного репортингу про проведені результати тестування є важливим елементом фреймворку автоматизації. Дані, наведені у звіті, використовуються для проведення аналізу на відповідність проведеної процедури до вимог та тест-дизайну, з метою запобігання невалідних верифікацій. Також, деталізований механізм репортингу є важливим для ефективного виявлення та дослідження виявлених дефектів у процесі

тестування.

Забезпечення декомпозиції тестових даних, та статичних змінних (констант) від методів всередині функціональних компонентів має значний вплив на підтримку і стабільність середовища виконання тестів та дозволяє скоротити час на підтримку, дозволяє уникнути дублювання коду та мінімізувати кількість несподіваних помилок.

Огляд основних елементів сучасного автоматизованого фреймворку для тестування адаптивних вебзастосунків дозволяє дійти висновку, що забезпечення належної якості автоматизованого тестування є багатогранним процесом, який виходить за межі написання скриптів, та потребує комплексного підходу, який відповідає потребам і вимогам бізнесу. Наразі, існуючими фреймворками надається базовий функціонал, призначений для взаємодії з вебелементами, а реалізація основних елементів фреймворку покладається на розробників.

Створення моделі масштабованої архітектури фреймворку автоматизації, який реалізує у собі функціонал основних архітектурних компонентів, є необхідним кроком, який дозволить тестувальникам зменшити часові витрати на впровадження і початок автоматизованого тестування на проєкті.

Висновки до розділу 1

Оптимізація процесів розробки програмного забезпечення потребує постійних змін, зокрема, у процесах тестування. Актуальність дослідження полягає у створенні моделі масштабованої архітектури фреймворку автоматизації тестування адаптивних веб-застосунків, яка дозволить зменшити вартісні та часові витрати на впровадження автоматизації.

Розглянуто особливості архітектури сучасних вебсистем, проаналізовано сучасні тенденції в автоматизації тестування та обґрунтовано доцільність його впровадження. У зв’язку зі зростанням кількості шляхів взаємодії між компонентами системи пропорційно збільшується час

проведення тестування. Впровадження автоматизації тестування є необхідним кроком, який дозволяє зменшити кількість людських помилок та час виконання тестів, зосередитись на забезпезпеченні належної якості об’єкту тестування та написанні ефективних тестових сценаріїв.

Проаналізовано та визначено переваги та недоліки сучасних патернів автоматизації, серед яких “Page object model”, “Behavior driven development”, “Keyword driven testing”. Обґрунтовано перспективи та потенціал реалізації комбінованих підходів, з метою підкреслення їх переваг та зменшення впливу недоліків у тестуванні.

Розглянуто основні елементи фреймворку автоматизації, який включає HTTP-клієнт, механізми інформативного репортингу, структуризацію та розподіл кодової бази. У зв’язку з тим, що існуючі фреймворки надають доступ до API та функціонал для взаємодії з елементами вебсторінок, а планування і проєктування фреймворку покладається на розробників – створення моделі масштабованої архітектури для тестування адаптивних веб-застосунків є необхідним кроком, який дозволить тестувальникам зменшити часові витрати на налаштування тестового середовища та адаптувати найкращі практики для потреб конкретного проєкту.

РОЗДІЛ 2. АНАЛІЗ ФРЕЙМВОРКІВ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ

2.1. Порівняльна характеристика популярних фреймворків

Формування і становлення багатофункціональних і ефективних фреймворків автоматизації тестування є результатом тривалого етапу становлення і модернізації застосунків. Від створення перших інструментів для написання юніт-тестів, до першого інструменту автоматизованого тестування у 2004 році інженерам програмного забезпечення необхідно було пройти ряд випробувань та технічних обмежень. Фреймворк “Selenium Core” став першим, який дозволив проводити автоматизоване тестування вебзастосунків та став основою для формування таких фреймворків, як Selenium, Cypress, Playwright, Cucumber та інших [20].

Порівняльна характеристика популярних фреймворків автоматизації тестування веб-застосунків демонструє етапи історичного розвитку автоматизованого тестування. Еволюція інструментів дозволяє визначити виклики, які сприяли виникненню нових фреймворків та способи подолання даних викликів. Особливості застосування та детальний огляд архітектури фреймворків Selenium та Cypress наведено у додатку Б.

2.2. Комплексний аналіз фреймворку Playwright

Playwright – фреймворк автоматизації End-to-end вебзастосунків, який підтримується та розвивається за підтримки Microsoft. Завдяки фінансовій і технічній підтримці даний інструмент підходить для тестування односторінкових вебзастосунків та адаптивних вебзастосунків. Також, фреймворк містить в собі великий набір вбудованих функцій, підтримку сучасних браузерів (Chromium, Webkit, Firefox) та підтримує написання коду за допомогою Javascript/Typescript, Java, Python, C#.

Playwright підтримує функції призначені для емуляції стандартних мобільних пристроїв та пристроїв, які мають нестандартне розширення екрану за допомогою можливості керування параметрами viewport. Слід відзначити,

що фреймворк має можливість виконувати автоматизовані тести у headless та headful режимах, підтримує повний цикл автоматизації тестування, включаючи в себе:

- інструменти для написання та відлагодження тестів;
- функціонал для формування звітності;
- інтеграція з CI/CD сервісами.

Відмінною особливістю архітектури Playwright від інших фреймворків є використання комунікації з драйверами браузерів за допомогою веб-сокет протоколу, на відміну від HTTP у Selenium та Cypress (рис. 2.1).

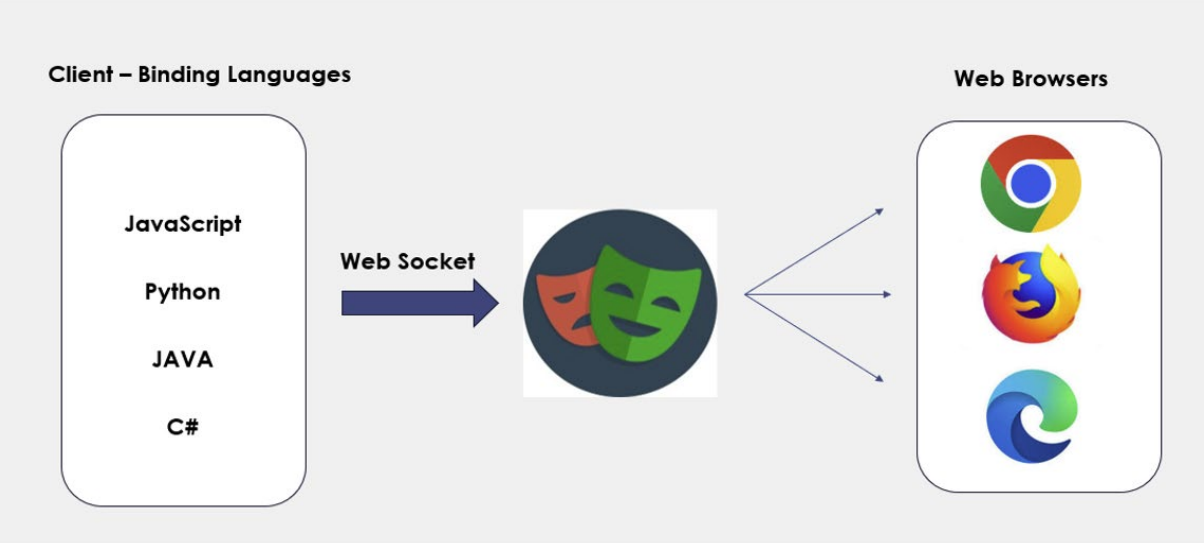


Рис. 2.1. Архітектура фреймворку Playwright

Джерело: Architecture Breakdown: Selenium, Cypress, and Playwright. Режим доступу: https://medium.com/@akssingh002/cypress-architecture-a-detailed-exploration-with-real-time-examples-d5cbf02b1030

Такий підхід до архітектури надає можливість для проведення комплексного процесу автоматизованого тестування, який включає в себе ізоляцію контексту та паралельне виконання тестів, гнучкість налаштування запуску необхідного набору тестів з визначеною конфігурацією.

Проведення аналізу функціональних особливостей фреймворку автоматизації Playwright дозволяє визначити ряд переваг та недоліків, з метою визначення доцільності його використання для автоматизації тестування адаптивних вебзастосунків.

Серед переваг Playwright відзначаються:

1. Кросплатформність.
2. Кросбраузерність та повноцінна адаптивність – підтримка браузерів Chrome, Edge, Firefox, Safari.
3. Перехоплення мережевих запитів – дозволяють здійснювати API тестування, або використовувати заглушки відповідей від серверу для здійснення UI тестування.
4. Динамічні очікування – можливість налаштування очікувань для появи елементів та подій. Дозволяє змінювати очікування за замовчуванням, так і для окремих елементів.
5. Підтримка багатопоточного виконання тестів – одночасне виконання автоматизованих тестів, за рахунок використання ресурсів системи.
6. Вбудована звітність – генерація HTML, текстового звітів по закінченню виконання тестів.
7. Можливість створення скріншотів та відео.
8. Вбудований trace viewer – дозволяє здійснювати відлагодження тесту та відстежувати його покрокове виконання.
9. Codegen – автоматичний генератор тестів. Працює шляхом запису дій користувача в браузері та генерує автоматизований сценарій тесту.

Недоліки фреймворку Playwright:

1. Використання системних ресурсів – багатопотокове виконання автоматизованих тестів, використання кількох браузерних сутностей використовують значні ресурси процесору і пам’яті.
2. Підтримка та екосистема – оскільки підтримка здійснюється компанією Microsoft, то необхідні оновлення і зміни відбуваються з більшими затримками часу, що може створювати проблеми при міграції на новіші версії фреймворку.
3. Складність налагодження – забезпечення адаптивного тестування системи потребує окремих налаштувань інженерами.
4. Розміри проєкту – оскільки фреймворк включає в себе рушії

браузерів, то фреймворк за замовчуванням використовує більше пам’яті на комп’ютері.

5. Вбудований HTTP клієнт – використання вбудованого HTTP клієнту є гарним рішенням для використання у поодиноких сценаріях. Коли є необхідність проведення повноцінного API тестування, можливості вбудованого HTTP клієнту є обмеженими та ускладнюють підтримку фреймворку.

6. Обмежена звітність – використання вбудованої звітності не є інформативним для повноцінних і складних сценаріїв, через що ускладнюється аналіз і локалізація дефектів та помилок тестових скриптів.

7. Відсутність підтримки застарілих версій браузерів.

Відзначивши відповідні переваги і недоліки фреймворку можемо дійти висновків, що Playwright є найбільш оптимальним фреймворком, який призначений для тестування адаптивних вебзастосунків. Повноцінні багатопотокове виконання тестових сценаріїв та адаптивність, можливість емуляції роботи мобільних пристроїв є основними перевагами інструменту. Проте обмежені HTTP клієнт, звітність та складність налагодження проєкту для повноцінного використання його функціональних можливостей у повному обсязі. Таким чином, для забезпечення якісних і ефективних процесів тестування необхідно здійснювати детальне проєктування архітектури фреймворку автоматизації та забезпечити механізми мінімізації наявних недоліків інструменту Playwright.

2.3. Вибір фреймворку автоматизації: можливості та виклики

Побудова фреймворку автоматизації є важливим і ресурсомістким етапом проєкту. Використання існуючих фреймворків автоматизації надають свої рішення у вигляді інструменту, над яким будується фреймворк автоматизації, проте цього недостатньо для побудови ефективної автоматизації тестування проєкту. Створені інструменти дозволяють використати дані рішення для обходу технічних обмежень, проте не надають

рішень для вдосконалення та покращення ефективності відповідних процесів тестування.

Обрання інструменту для автоматизації тестування є основою для побудови ефективних процесів. Аналіз кількості існуючих технічних обмежень та наявних функціональних можливостей надає вичерпну інформацію, яка дозволяє оцінити об’єми та приблизні вартісні та часові витрати на реалізацію. У розрізі сучасних адаптивних вебзастосунків виклики пов’язані з обмеженою підтримкою кросбраузерного тестування, відсутністю паралельного виконання тестів, надання інформативної звітності про результати виконання тестового циклу представляють значні ризики ефективності і доцільності впровадження автоматизованого тестування на проєкті. Саме тому, побудова архітектури фреймворку автоматизації адаптивних вебзастосунків повинна включати в себе наступні функціональні можливості:

- можливість емуляції роботи мобільних пристроїв (дії tap, swipe, tap-and-hold);
- підтримка паралельного виконання тестів;
- підтримка кросбраузерності;
- наявність HTTP-клієнту для відправки API запитів;
- наявність інструментів для стабілізації тестів та дебаггінгу;
- створення інформативних звітів результатів виконання тесту;
- здатність до функціонального масштабування – можливість використання додаткових пакетів і бібліотек, які можуть бути необхідними для вирішення певних проблем тестування, які є характерними для даного проєкту.

Проведений аналіз існуючих фреймворків надає можливість визначення характерних особливостей, які дозволяють визначити інструмент, використання якого є найбільш доцільним для тестування адаптивних вебзастосунків. У роботі розглянуто функціональні можливості та особливості архітектури фреймворків Selenium, Cypress та Playwright. Проте, окрім

обрання фреймворку для створення середовища тестування не менш важливими є наступні фактори:

- обрання доцільного для використання патерну автоматизації (Page Object, Page factory);
- логічний розподіл елементів тестового середовища (константи, утиліти, тестові дані, конфігураційні файли, тестові скрипти);
- визначення загальних командних підходів до автоматизації (єдиний стиль написання коду, алгоритм внесення змін до фінальної версії автоматизованого фреймворку);
- інтеграція з CI/CD системами;
- використання принципів SOLID, DRY, KISS, які спрощують підтримку кодової бази проєкту.

Використання перерахованих принципів не враховується в існуючих фреймворках, але їх використання є важливим для побудови прозорого та ефективного середовища тестування. Організація формалізованої командної взаємодії, використання встановлених правил рев'ю коду, підходів до оформлення коду та документування автоматизованих тестів дозволяють забезпечити прозорість процесів тестування та дозволяє спростити адаптацію нових членів команди на проєкті. Впровадження і використання даних факторів дозволяє створити ефективне тестове середовище, яке дозволяє забезпечити не лише доступність його використання, але й здатність до масштабування та точність і якість тестування застосунків.

2.4. Переваги використання фреймворку Playwright у контексті адаптивних вебзастосунків

Проведений аналіз популярних фреймворків автоматизації тестування, таких як Selenium, Cypress, Playwright. Огляд архітектури і функціональних особливостей дозволяють визначити, що серед наведених фреймворків Playwright має повноцінну підтримку адаптивності та емуляцію мобільних пристроїв.

Тестування адаптивних вебзастосунків має великий перелік особливостей, де застосування класичних підходів тестування неможливе. Здійснення кліків по елементах вебсторінок, позиціонування елементів та їх доступність, структура DOM дерева відрізняються для застосунків, які виконуються на мобільних пристроях та планшетах. Натомість, можливість здійснення емуляції дій, які виконуються на сенсорних пристроях (tap, swipe, drag-and-drop, pinch-zoom) дозволяють створювати повноцінні та ефективні автоматизовані тестові скрипти.

Використання методів, які дозволяють взаємодіяти та імітувати поведінку застосунку на мобільних пристроях – важливий крок, який наближає автоматизацію тестування до досягнення цілей тестування. Проте, це є неможливим без забезпечення функціональності, яка дозволяє здійснювати верифікації та визначати очікувані результати поведінки системи після виконаних користувачем дій. Важливими є перевірки видимості елементів, коректне відображення їх розмірів (елементи не виходять за визначене значення розмірів viewport), перевірки, які дозволяють визначати зміну стану елементів інтерфейсу після настання подій та інші.

Досягнення адаптивних розмірів viewport браузерів, емуляції роботи мобільних пристроїв є значним викликом для фреймворків Cypress та Selenium. Це є однією з причин, чому автоматизація адаптивних веб-застосунків залишає прогалини та має обмежений перелік функціональності, яка може бути перевірена автоматизованим способом. Натомість, все більшого поширення на сучасному ринку займає фреймворк Playwright. Серед його доступних функціональних особливостей у тестуванні адаптивності відзначаються:

- динамічне керування розмірами вікна браузера;
- емуляція функцій tap, swipe, геолокації та інших специфічних функцій, характерних для мобільних пристроїв;
- тестування доступності;
- можливість емулювати затримки мережі, що дозволяє тестувати

сценарії, пов’язані з низькою швидкістю Інтернету у користувачів;

- набір методів для перевірки зміни станів окремих елементів та динамічних очікувань;
- емуляція пристроїв, які використовують операційні системи Android та iOS;
- гнучкість налаштування запуску наборів тестів з конфігурацією під визначену платформу;
- паралельне виконання тестів та ізольованість контексту.

Впровадження ефективної автоматизації тестування проєкту значною мірою залежить від обрання ключового інструменту, навколо якого будується архітектура фреймворку автоматизації тестування. Визначення підходів до автоматизації, забезпечення структурованої кодової бази, констант, інтеграції з інструментами інформативного репортингу (Allure, JUnit), CI/CD сервісами, бібліотеками, які частково вирішують проблеми підтримки тестового середовища або дозволяють вирішувати типові проблеми, які виникають під час підтримки фреймворку автоматизації.

Відповідно, до визначених технічних особливостей і викликів, пов’язаних з тестуванням адаптивних вебзастосунків фреймворк Playwright є комплексним інструментом, який відмінно підходить для використання у вигляді ядра для побудови масштабованого та ефективного фреймворку автоматизації тестування.

Висновки до розділу 2

Selenium є першим фреймворком автоматизації веб-застосунків, який взаємодіє з браузером за допомогою перетворення HTTP-запитів у команди які виконуються браузером.

Розвиток користувальницького інтерфейсу і поширення вебзастоунків з клієнт-серверною архітектурою створило нові виклики тестування. Як відповідь на існуючі обмеження, виникнення фреймворку Cypress стало наступним проривом у автоматизації, після Selenium. Особливими перевагами

даного фреймворку є можливість імітації відповідей від серверу та можливість перехоплення мережевих запитів. Проте, важливим недоліком даного інструменту є відсутність емуляції мобільних пристроїв та належної підтримки кросбраузерності.

Значний приріст популярності використання вебзастосунків на мобільних пристроях сприяло створення нового фреймворку Playwright, який використовує новий підхід взаємодії з браузером на основі веб-сокетів. Playwright є інструментом, який спрямований на тестування адаптивних вебзастосунків. Наявність емуляції мобільних пристроїв, можливість багатопотокового виконання тестів, гнучкість налаштування конфігураційних файлів, наявність інструментів для відлагодження тестів роблять Playwright потужним інструментом для побудови ефективного і масштабованого середовища тестування адаптивних вебзастосунків.

РОЗДІЛ 3. РОЗРОБКА МОДЕЛІ МАСШТАБОВАНОЇ АРХІТЕКТУРИ ФРЕЙМВОРКУ АВТОМАТИЗАЦІЇ

3.1. Постановка завдання: визначення моделі та вимог до архітектури

Визначення моделі масштабованої архітектури фреймворку автоматизації тестування адаптивних вебзастосунків передбачає логічну структуру системи та її фізичну реалізацію, що дозволяє забезпечити впровадження і підтримку автоматизованого тестування у нові та вже існуючі проєкти. Модель фреймворку автоматизації повинна бути реалізована у відповідності до загальноприйнятого життєвого циклу автоматизації тестування.

Модель побудована у відповідності до життєвого циклу автоматизації тестування дозволяє отримувати достовірні результати тестування, економити час виконання циклів тестування [3]. Отримані формалізовані та інформативні звіти, надають можливість здійснювати оцінку готовності програмного продукту до випуску та приймати рішення стосовно подальшої розробки усім зацікавленим сторонам [28].

Відповідно до тематики і завдань кваліфікаційної роботи магістерського рівня визначено наступні вимоги до моделі фреймворку автоматизації тестування адаптивних вебзастосунків:

1. Реалізувати архітектуру фреймворку автоматизації відповідно до загальної архітектури автоматизації з використанням інструменту для автоматизації тестування Playwright.
2. Забезпечити рівневу архітектуру (логічний розподіл між тестовими скриптами, бізнес-логікою та сервісними бібліотеками).
3. Забезпечити здатність до масштабування.
4. Використати принципи програмування та дизайн-патерни, які дозволяють зменшити часові витрати на підтримку фреймворку автоматизації та забезпечити прозорість підходів для роботи команди.
5. Забезпечити деталізовану звітність та логування результатів

виконання автоматизованих тестів.

6. Організувати конфігурацію фреймворку для виконання тестів для desktop, tablet та mobile пристроїв.

7. Запровадити механізми для інтеграції з CI/CD.

8. Імплементувати мінімально необхідний функціонал бібліотек, які застосовуються для тестування адаптивних вебзастосунків.

9. Передбачити можливість паралельного виконання автоматизованих тестів.

Відповідно до рекомендацій International Software Testing Qualifications Board запропоновано рівневу модель архітектури фреймворку автоматизації адаптивних веб-застосунків, яку розділено на 4 основні рівні (рис. 3.1) [29].

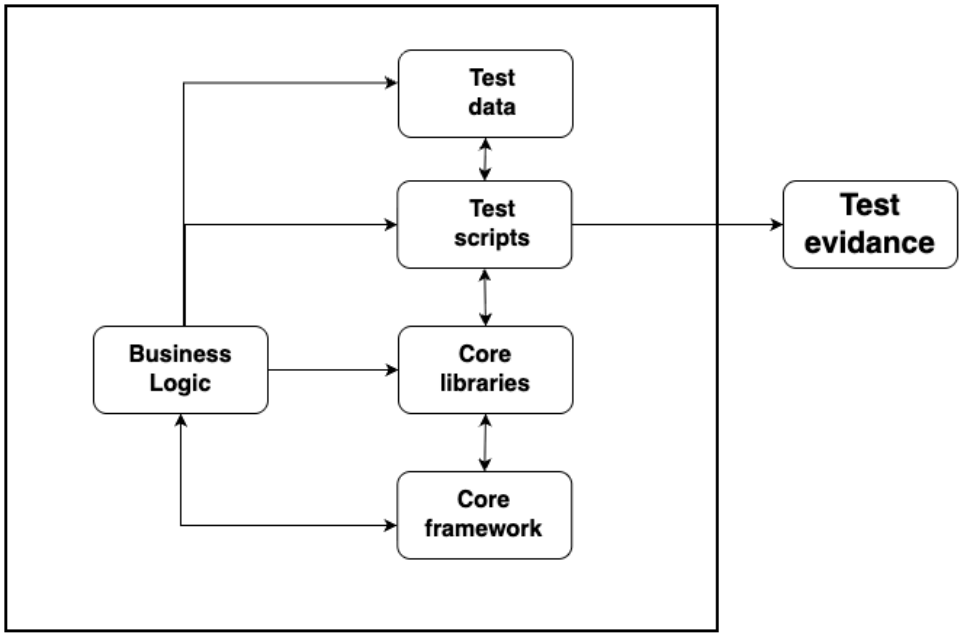


Рис. 3.1. Рівні архітектури моделі фреймворку

Перший рівень (Core framework) моделі представлений набором методів та функцій фреймворку Playwright для взаємодії з браузером та запуском автоматизованих тестів. Обраний фреймворк повинен надавати необхідний перелік функціональності, для реалізації бізнес-логіки роботи застосунку.

Другий рівень моделі (Core libraries) – поєднання рівня бізнес-логіки застосунку та базових бібліотек, що описують основні способи взаємодії з адаптивним вебзастосунком. Об’єднання шару бізнес логіки та базових бібліотек в один рівень моделі обґрунтовано тим, що шар бізнес-логіки

визначає структуру і вміст набору основних бібліотек. Додатково, реалізація основних бібліотек відповідно до потреб бізнес-логіки спрощує підтримку та адаптацію нових учасників проєкту, оскільки вся архітектура моделі створюється з урахуванням єдиного підходу. Шар бізнес-логіки на даному рівні моделі може бути представлений використанням загальноприйнятих назв для функцій, які є зрозумілими для усіх спеціалістів, залучених до реалізації проєкту.

Третій рівень (Test scripts) – рівень автоматизованих тестових скриптів. На даному рівні визначається набір тестових сценаріїв, що здійснюють безпосередні дії та перевірки об’єкту тестування на відповідність вимогам. Даний рівень повинен приховувати аспекти технічної реалізації, яка представлена на рівні основних бібліотек. Описані тестові сценарії повинні мати необхідний рівень абстракції, щоб за результатами виконання тестів інженери контролю якості могли локалізувати причини неуспішного виконання тестів (дефект об’єкту тестування чи проблема реалізації автоматизованого скрипта/бібліотеки). Рівень автоматизованих тест-скриптів є заключним кроком у роботі моделі фреймворку автоматизації, тому після виконання циклу тестування повинен бути автоматизовано створений звіт про результати тестування.

Верхній рівень (Test data) – рівень тестових даних моделі представлений тестовими даними. Відповідні структури даних та їх вміст визначається відповідно до бізнес-логіки роботи застосунку та вимог до ПЗ, що розробляється.

Вміст другого рівня моделі (Core libraries) визначається трьома основними пакетами:

1. Page-Object library – бібліотека, що описує структуру і поведінку усіх сторінок адаптивного веб-застосунку.

2. Api-Client – бібліотека, яка описує структуру запитів і відповідей до серверу. В основі лежить клас-обгортка, який дозволяє стандартизувати конфігурування запитів у всьому проєкті. Даний клас призначений для

спрощення підтримки фреймворку.

3. Utils – набір бібліотек, який містить набір допоміжних класів і бібліотек, які необхідні для проведення тестування. У даному пакеті визначаються функції і методи для роботи з датами, обчисленнями, генерацією та перетвореннями визначених даних. Також, утиліти можуть містити в собі необхідні конфігураційні файли.

Структура другого рівня моделі – “Core libraries” представлена на рисунку 3.2.

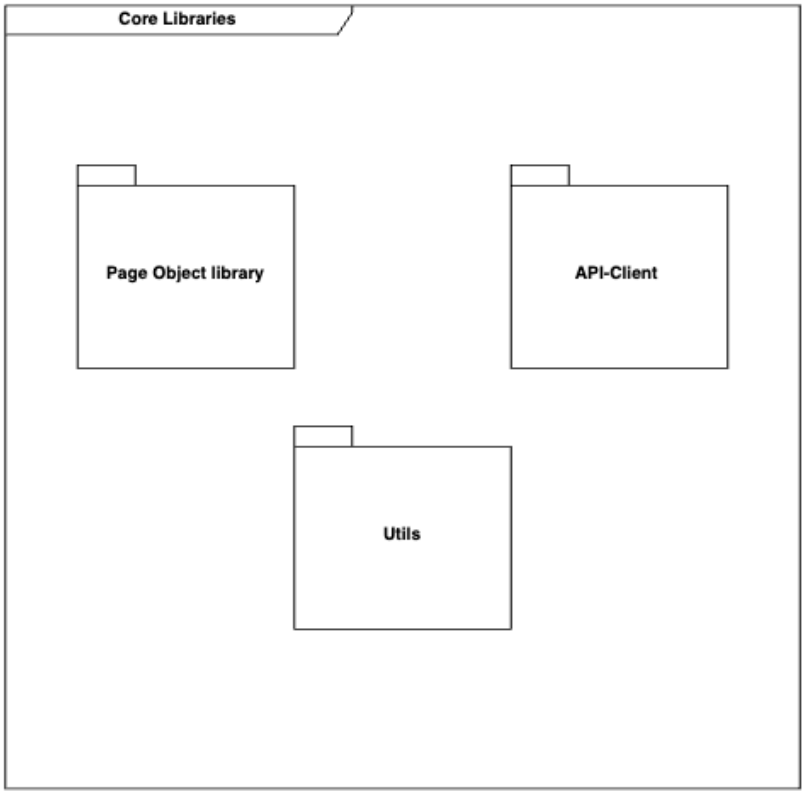


Рис. 3.2. Вміст Core libraries – другого рівня моделі фреймворку

Використання розподілу фреймворку автоматизації тестування адаптивних вебзастосунків на чотири рівні, серед яких тестові дані виокремлюються як окремий рівень, а рівень бізнес-логіки взаємодіє з усіма рівнями моделі обґрунтовується тим, що адаптивні вебзастосунки, у більшості випадків створюються відповідно до гнучких методологій розробки програмного забезпечення, таких як Scrum або Kanban. Зважаючи на те, що впродовж ітераційних циклів розробки ПЗ вимоги до програмного продукту можуть часто змінюватись, то реалізація взаємодії шару бізнес логіки з усіма рівнями моделі фреймворку дозволяє виявити деякі дефекти у вимогах до

програмного продукту. Таким чином стає можливим виявлення дефектів у ПЗ, ще до моменту кодової реалізації та, як наслідок, зниження вартості виправлення дефектів [30].

Повну модель масштабованої архітектури фреймворку автоматизації тестування адаптивних вебзастосунків представлено на рисунку 3.3.

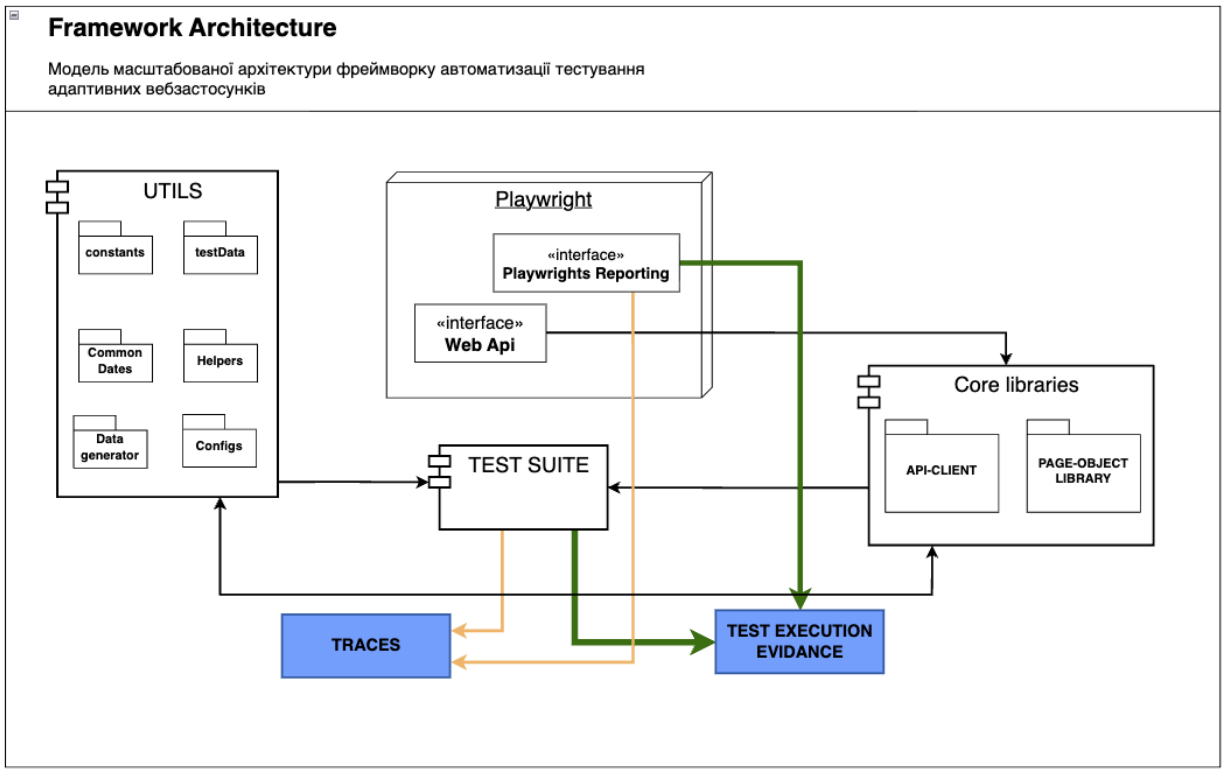


Рис. 3.3. Модель масштабованої архітектури фреймворку автоматизації тестування адаптивних вебзастосунків

Представлена архітектура відповідає визначеним рівням моделі та демонструє зв'язки між окремими компонентами моделі. Результатом проведення циклу автоматизованого тестування є генерація звіту про результати виконання тестів та генерація traces-файлів, що відображають покрокове виконання тесту. Traces використовуються для локалізації дефектів та під час створення автоматизованих тестів.

3.2. Реалізація модульності та гнучкості моделі архітектури (імпорти, налаштування, конфігураційні файли)

Використання фреймворку Playwright надає необхідний функціонал для тестування адаптивних вебзастосунків. Проте, структура проєкту за

замовчуванням містить конфігураційний файл інструменту, файл маніфест проєкту Node.js, директорію для зберігання автоматизованих тестів та папку node_modules, де знаходяться всі встановлені залежності проєкту (рис. 3.4).

Рис. 3.4. Структура проєкту (за замовчування) з використанням Playwright

Як видно на рисунку 3.4 проєкт створений за допомогою фреймворку Playwright.

З метою дотримання рівневої архітектури моделі фреймворку файлову структуру проєкту розподілено на 3 основні категорії:

1. Core libraries – усі відповідні файли і бібліотеки зберігаються у директорії “scr”.

2. Test scripts – зберігання тестових сценаріїв окремо від іншої кодової реалізації моделі фреймворку.

3. Конфігураційні файли – файли проєкту верхнього рівня, які не належать до тестових скриптів та набору бібліотек, які використовуються для проведення тестування.

Додатково, після виконання циклу тестування автоматичного генерується директорія playwright-report, яка містить звіт про виконання тестів.

Файлову структуру верхнього рівня наведено на рисунку 3.5.

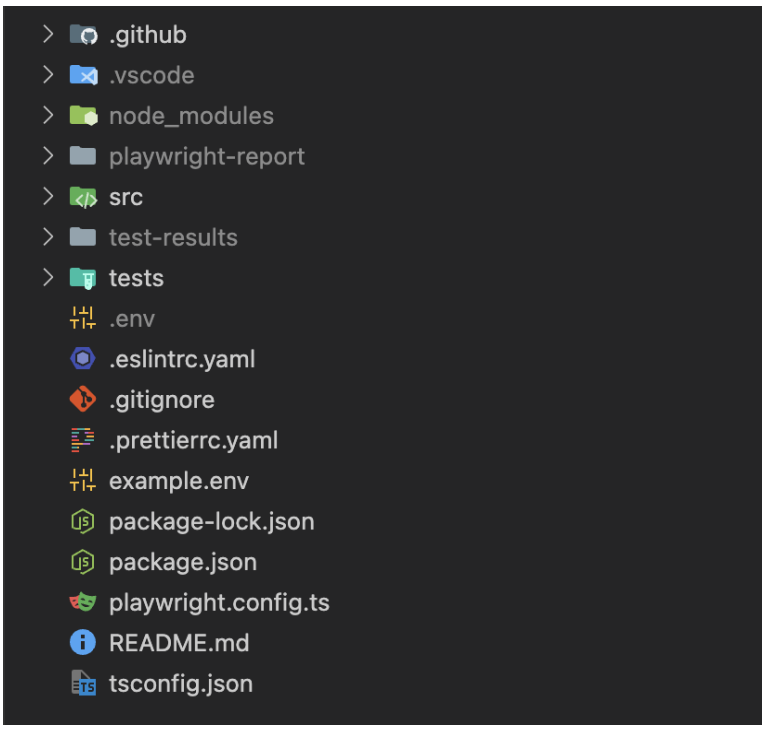


Рис. 3.5. Файлова структура верхнього рівня моделі архітектури

Забезпечення модульної архітектури файлів досягається шляхом визначення певних параметрів у конфігураційних файлах проєкту. Для того, щоб ініціалізувати модульну структуру необхідно у файлі package.json визначити відповідне поле у форматі "type":"module". Наступним кроком у файлі tsconfig.json доцільно використати аннотації, які дозволяють скоротити шляхи імпортів. Також дане налаштування є корисним для підтримання файлової структури проєкту та спрощує навігацію у проєкті та його підтримку (рис 3.6).

```
26    /* Modules */
27    "module": "commonjs" /* Specify what module code is g
28    "rootDir": "./" /* Specify the root folder within you
29    // "moduleResolution": "node10",
30    "baseUrl": "./" /* Specify the base directory to reso
31    "paths": {
32      "@testExtenter": ["src/pages/testExtender.ts"],
33      "@pages/*": ["src/pages/*"],
34      "@types/*": ["src/types"],
35      "@utils/*": ["src/utils/*"],
36      "@helpers/*": ["src/utils/helpers/*"],
37      "@interfaces/*": ["scr/interfaces/*"]
38    } /* Specify a set of entries that re-map imports to
```

Рис. 3.6. Налаштування імпортування модулів у tsconfig.json файлі

Приклад використання імпортів під час написання коду на рисунку 3.7.

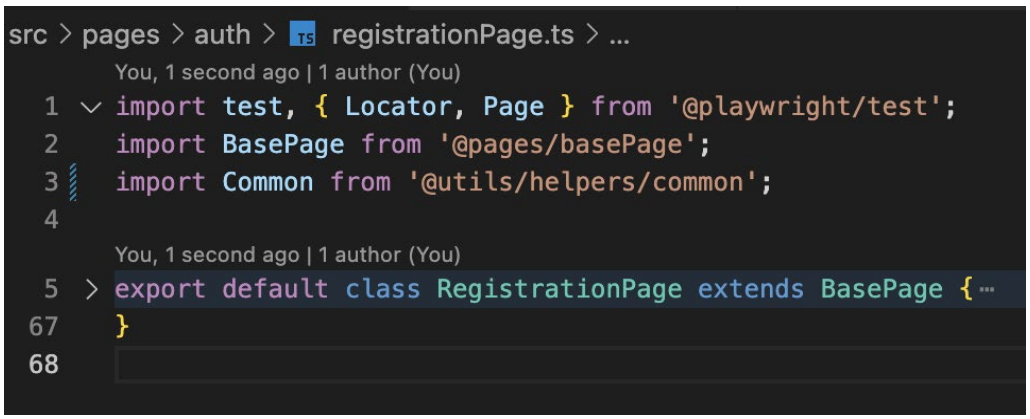


Рис. 3.7. Використання анотацій імпорту для використання у коді

Використання імпортів у такому вигляді дозволяє технічним спеціалістам швидко здійснювати навігацію у проєкті та чітко визначати пов'язані між собою файли. З метою забезпечення модульності гарною практикою є використання окремого файлу з назвою .env, який містить в собі чутливі дані, такі як паролі доступу до баз даних, налаштування оточення, адреси серверів та ключі до API, які використовуються для тестування адаптивних вебзастосунків.

3.3. Визначення основних методів для роботи з API, датами, часовими зонами

Для зручності роботи з API замість використання вбудованих інструментів Playwright було прийнято рішення використовувати зовнішню бібліотеку axios, оскільки вона має певний перелік переваг, серед яких:

1. Використання зрозумілого синтаксису та наявність детальної документації.
2. Можливість перехоплення HTTP-запитів.
3. Автоматичне перетворення даних в JSON формат.
4. Можливість перетворення даних запиту та відповіді від серверу [31].

З метою мінімізації залежностей від змін у вихідному коді реалізовано клас обгортку, який використовується для виконання всіх HTTP-запитів під час проведення тестування.

Даний клас представлений у моделі за шляхом "src/utils/api" з назвою ApiClient. У даному класі реалізовано наступні методи: get, post, put, delete,

getAuthToken – для отримання токену авторизації користувача.

Даний клас може використовуватись для API-тестування, а також для виконання перед- та пост-умов необхідних для виконання тестових сценаріїв. Коли передбачається використання наборів даних різної природи, то зазвичай, створюються класи, які описують відповідну схему даних та перелік доступних дій з ними. Приклад використання класу ApiClient для створення користувачів у вебзастосунку наведено на рисунку 3.8.

Рис. 3.8. Приклад використання класу ApiClient у коді проєкту

Проведення операцій з даними на сервері та їх відображенням на інтерфейсі у вебзастосунках є важливим етапом роботи системи. Відповідно, виникає потреба контролювати і відстежувати зміни стану даних на сервері. Саме тому виникає потреба роботи з різними часовими форматами та часовими зонами. З метою реалізації функціоналу для роботи з часовими зонами та різними форматами часу визначено:

- файл base_const – визначає основні існуючі формати даних;
- класс Common – визначає основні операції з датами, серед яких: отримання поточної дати, методи для форматування дати з урахуванням таймзон, отримання різниці між датами, конвертування дат з одного формату в інший.

Робота з часовими зонами реалізована за допомогою використання бібліотеки “Luxon”, яка надає API для отримання поточного часу у відповідній

локації з динамічним урахуванням переходу між літнім та зимовим часом.

3.4. Побудова архітектури на основі Page Object Model

Для опису вмісту сторінок та доступних дій з ними було обрано структурний патерн автоматизації Page Object Model. В основі даного патерну лежить опис базової сторінки, яка містить перелік функцій, які є доступними для всіх сторінок застосунку. Даний патерн має високу здатність до масштабування та є простим для розуміння.

У моделі масштабованої архітектури фреймворку автоматизації тестування адаптивних вебзастосунків опис об'єкту тестування визначається у “Page Object library”. Відповідно, у файловій структурі фреймворку дана бібліотека представлена за шляхом src/pages (див. Додаток Г).

Першим кроком реалізації підходу Page Object є створення базової сторінки, яка надає базовий екземпляр контексту сторінки браузеру – page, який наслідується усіма дочірніми класами. BasePage клас містить наступний перелік методів: waitUntilLoad, goTo, verifyPageURL, verifyPageTitle – здійснює перевірку заголовку відповідної сторінки, clearAllCookies – видаляє усі наявні файли cookies. Структурно реалізація бібліотеки сторінок визначається за відповідним зразком наведеним на рисунку 3.9.

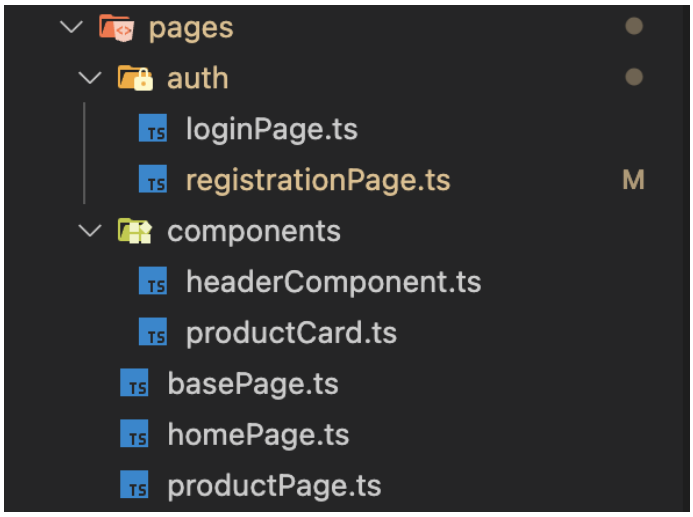


Рис. 3.9. Файлова структура Page Object libraries

Відповідно до визначеної структури сторінки повинні описуватись в одному з трьох форматів:

1. Feature pages – сторінки, які належать до частини додатку, яка виконує певну функцію. Наприклад, сторінки входу і реєстрації виконують функцію автоматизації.

2. Components – визначення окремих компонентів сторінок, які можуть використовуватись на кількох сторінках додатку.

3. Загальні – сторінки, які не поєднані з іншими сторінками функціонально.

Усі типи сторінок вебзастосунку повинні успадковуватись від базового класу BasePage. Додатково, для використання описаних сторінок та доступної функціональності використовується файл testExstender. Даний файл описує фікстури, щоб зробити сторінки доступними для використання у тестах без використання класичних імпортів. За допомогою testExtender ми розширюємо контекст фікстури test. Приклад файлу testExtender наведено на рисунку 3.10.

```
export const test = base.extend<MyPages>({
  homePage: async ({ page, isMobile }, use) => await use(new HomePage(page, isMobile)),
  loginPage: async ({ page, isMobile }, use) => await use(new LoginPage(page, isMobile)),
  registrationPage: async ({ page, isMobile }, use) => await use(new RegistrationPage(page, isMobile)),
  headerComponent: async ({ page, isMobile }, use) => await use(new HeaderComponent(page, isMobile)),
  productPage: async ({ page }, use) => await use(new ProductPage(page)),
});
```

Рис. 3.10. Використання testExtender у Page Object libraries

У файлі зі скриптом тесту використання сторінок відбувається як показано на рисунку 3.11.

```
Run | Debug
test(`Create new user with valid credentials`, async ({ homePage, registrationPage, headerComponent, browserName }) => {
  Run | Debug
  test.skip(browserName === 'firefox', 'Failed by defect');      You, 13 months ago • skipped test "Create new user with va
  const registrationData = fakerDataGenerator.generateNewUserData();
  await homePage.clickOnCreateAnAccountLink();
  await registrationPage.verifyPageURL('customer/account/create/');
  await registrationPage.fillAndSubmitRegistrationForm(registrationData);
  await headerComponent.verifyLoggedInMessage(registrationData.firstname + ' ' + registrationData.lastname);
});
```

Рис. 3.11. Використання сторінок testExtender у Test Scripts

Даний підхід дозволяє позбутись надлишкових імпортів у тестовому файлі та спрощує підтримку тестів.

3.5. Скорочення часу виконання тестів та формування звітності про результати виконання автоматизованих тестів

3.5.1. Конфігурація для кросбраузерного тестування на різних платформах (mobile, tablet, desktop)

Конфігурація для виконання тестування для кросбраузерного тестування на різних типах пристроїв передбачає низку необхідних налаштувань. Першим етапом для реалізації адаптивності є налаштування і оновлення конфігураційного файлу playwright.config.json. У даному файлі необхідно визначити відповідні типи пристроїв, з розширенням екрану яких виконуватимуться тести. Приклад визначення конфігурації для емуляції роботи на мобільному пристрої наведено на рисунку 3.12.

Рис. 3.12. Визначення різних типів пристроїв у playwright.config.json

Другим етапом підготовки тестового середовища до тестування різних типів пристроїв є оптимізація та оновлення класів, що описують сторінки адаптивного вебзастосунку. Необхідно визначити відповідну властивість isMobile у класі сторінки, адаптувати локатори елементів та здійснити рефакторинг методів відповідно до очікуваної поведінки системи, що тестується на мобільному пристрої. У файлі testExtender важливо передати параметр isMobile у конструкторі адаптивних веб-сторінок. Параметр isMobile

визначається відповідно до налаштувань проєкту у конфігураційному файлі.

Заключним етапом налаштування моделі для тестування з використанням конфігурацій для адаптивних пристроїв є налаштування скриптів, у файлі package.json. Дані скрипти необхідні для запуску відповідних тестів з використанням визначених конфігурацій (рис. 3.13).

```
"scripts": {
  "test api": "npx playwright test tests/examples/api --project=api",
  "test desktop": "npx playwright test tests/examples/ui  --project=Chrome --project=Firefox --project=Safari",
  "test desktop headed": "npx playwright test tests/examples/ui --project=Chrome --project=Firefox --project=Safari --headed",
  "test mobile": "npx playwright test tests/examples/ui --project='iPhone 12 Chrome'  --project='iPhone 12 Safari'",
  "test mobile headed": "npx playwright test tests/examples/ui --project='iPhone 12 Chrome'  --project='iPhone 12 Safari' --headed",
  "test tablet": "npx playwright test tests/examples/ui --project='Galaxy Tab S4 Chrome' --project='Galaxy Tab S4 Safari'  --project=
  "test tablet headed": "npx playwright test tests/examples/ui --project='Galaxy Tab S4 Chrome' --project='Galaxy Tab S4 Safari'  --pr
},
```

Рис. 3.13. Визначення скриптів для запуску тестів з встановленими конфігураціями у package.json

Реалізація підтримки кросбраузерного тестування на різних типах пристроїв є комплексним рішенням, яке повинно бути імплементоване на перших стадіях проєкту, оскільки потребує проведення рефакторингу кодової бази. Чим пізніше буде впроваджено підтримку адаптивності, тим більші об’єми робіт потребують виконання, внаслідок чого виростають часові та вартісні витрати на автоматизацію. Відповідно до рівневої моделі архітектури для впровадження адаптивності виникає необхідність рефакторингу коду лише на одному рівні – Core libraries.

3.5.2. Налаштування CI/CD процесів за допомогою GitHub Actions

З метою раціонального використання ресурсів логічною є реалізація виконання автоматизованих скриптів на віддаленому сервері, а саме за допомогою використання CI/CD процесів [6]. Для реалізації даної функціональності у моделі здійснено інтеграцію з сервісами Github Actions. Відповідно до потреб тестування адаптивних веб-застосунків створено три конфігураційні файли для запуску тестів на визначених типах пристроїв: desktop, mobile, tablet. Дані файли сконфігуровані так, щоб після внесення коду у головну гілку проєкту, виконання тестів відбувалось автоматично на відповідних платформах. Також, враховано, що виконання тестів може бути

розпочато мануально у Github репозиторії з використанням вкладки Actions. Таким чином, виконання тестів можна розпочати у будь-який момент, обравши одну чи декілька конфігурацій за відповідним типом пристрою.

3.5.3. Реалізація репортингу (Allure, HTML-звіти) та відлагодження тестів

Для отримання інформативного репортингу у моделі реалізовано формування звітів HTML за допомогою Playwright та Allure (див. Додаток Д). Для цього необхідно визначити репортери у файлі playwright.config.ts:

reporter: [['html', { open: 'never' }], ['list'], ['allure-playwright']],

після чого буде відбуватись автоматична генерація звітів після виконання тестів.

Приклад Allure звіту наведено на рисунку 3.14.

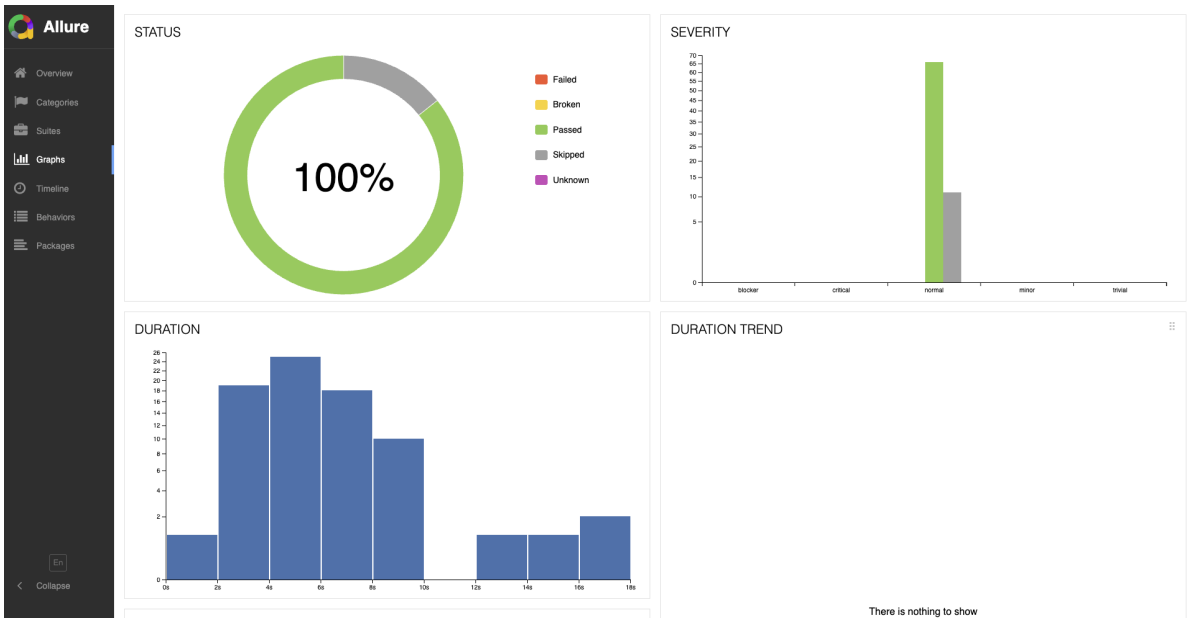


Рис. 3.14. Allure звітність реалізована у моделі архітектури.

Звітність Allure надає візуалізацію про виконання тестового циклу визначаючи процент успішно виконаних тестів, їх час виконання та групування за конфігураціями.

HTML-репортер Playwright має вигляд веб-сторінки, яка має секцію фільтрів, та переліку тестів з зазначенням конфігурації, часу та статусу виконання тестів (рис. 3.15).

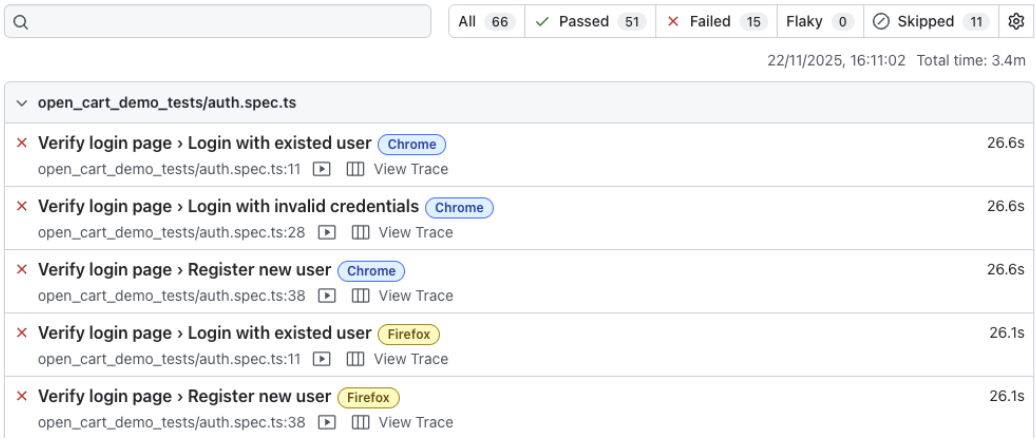


Рис. 3.15. HTML reporter Playwright.

Особливою перевагою використання HTML-репортера Playwright є наявність traces – детальної та покрокової інструкції виконання тестів. Даний механізм є корисним для відлагодження, стабілізації автоматизованих скриптів. У моделі реалізовано додавання скріншотів, відео та трейсів для тих автоматизованих скриптів, які завершились з неуспішним статусом виконання.

3.5.4. Підтримка паралельного виконання тестів та ізольованості контексту виконання

У моделі передбачено можливість виконання тестів у декілька потоків, яка досягається шляхом налаштування параметру workers у конфігураційному файлі playwright.config.ts. Даний параметр визначає скільки максимально може бути використано потоків під час виконання тестів. Під час тестування моделі виконання набору автоматизованих тестів виконувалось у 5 потоків (рис. 3.16)

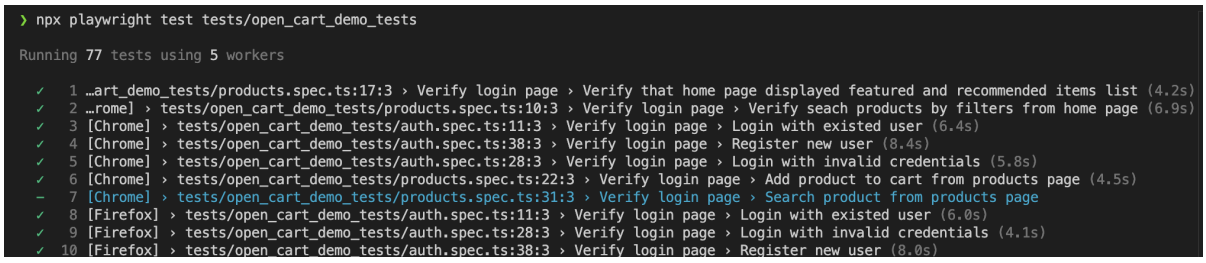


Рис. 3.16. Демонстрація багатопотокового виконання тестового набору для моделі.

Для того, щоб тестування було можливим для асинхронного виконання

та багатопоточно, необхідно дотримуватись принципів проєктування тестів та рекомендацій Playwright, щодо досягнення ізольованості контексту виконання [32].

Детальний опис тестування та проведення аналізу результатів тестування моделі масштабованої архітектури фреймворку автоматизації тестування адаптивних вебзастосунків наведено у додатку В.

Висновки до розділу 3

Визначено рівні моделі та побудовано схему архітектури фреймворку автоматизації відповідно до рекомендацій та підходів визначених ISTQB. Створено базову реалізацію та інтеграцію бібліотек необхідних для тестування адаптивних застосунків. Інтегровано використання Github Actions, забезпечено можливість виконання автоматизованих тестів у віддаленому репозиторії та у багатопотоковому режимі. Забезпечено механізми репортингу та traces за допомогою Playwright HTML-report та Allure report. Здійснено налаштування звітності з генерацією скріншотів, відео для тестів. Завершена модель є комплексним рішенням, яке дозволяє ефективно тестувати адаптивні веб-застосунки, з дотриманням сучасних рекомендацій і підходів, рекомендованих для впровадження автоматизованого тестування, без необхідності проведення передчасного рефакторингу, оскільки модель реалізовано з використанням clean-code підходів.

ВИСНОВКИ

Автоматизоване тестування у контексті адаптивних вебзастосунків є важливим кроком для успішної реалізації проєкту. Високі вимоги користувачів, необхідність мінімізувати ризики від впровадження системи у бізнес-процеси ставлять замовників, які потребують реалізації програмного продукту, у важку ситуацію, коли необхідно обирати між часом та вартістю. Дана проблема значною мірою підсвічується, високою конкуренцією на ринку програмного забезпечення. Впровадження ефективного автоматизованого тестування для адаптивних вебзастосунків вимагає високого рівня професійних навичок і глибокої експертизи від фахівців контролю якості та, відповідно, значних вартісних вкладень. Розроблена модель масштабованої архітектури фреймворку автоматизації тестування адаптивних вебзастосунків дозволяє зменшити вартісні витрати на впровадження автоматизованого тестування та забезпечити належну якість тестування програмного продукту.

У першому розділі розглянуто сучасні тенденції та підходи до тестування адаптивних вебзастосунків та існуючі підходи до автоматизації тестування. Продемонстровано важливість правильного підходу до впровадження автоматизації у проєкт та обґрунтовано важливість створення моделі, яка відповідає сучасним вимогам до тестування.

У другому розділі зазначено історичні аспекти виникнення інструментів для автоматизованого тестування вебзастосунків, особливості їх архітектури та визначення випадків, у яких є доцільним використання даних інструментів. Здійснено детальний огляд фреймворків Selenium, Cypress та Playwright. Обґрунтовано переваги використання Playwright у вигляді основного інструменту для побудови моделі, серед яких визначається використання комунікації з браузером за допомогою websocket, підтримка багатопоточності, емуляції розмірів мобільних пристроїв та унікальних дій з ними, таких як tap, swipe, drag-and-drop.

У третьому розділі визначено вимоги до моделі масштабованої архітектури фреймворку автоматизації тестування адаптивних вебзастосунків,

створено її рівневу схему та архітектуру. Визначено і обґрунтовано етапи досягнення модульності, гнучкості та масштабованості моделі. Створено набір та інтегровано перелік необхідних модулів і бібліотек, які забезпечують можливість роботи з часовими зонами та датами, виконання межевих запитів до серверу та інформативну звітність, яка є необхідною для виявлення дефектів і аналізу достовірності результатів тестування.

Успішні результати тестування моделі архітектури призначеної для тестування адаптивних вебзастосунків свідчать про її ефективність та здатність до масштабування. Впровадження даної моделі у комерційні проєкти потребує лише внесення коду, який описує об’єкт тестування та доступні дії для взаємодії з ним. Чітко розділена рівнева структура та модульність спрощують підтримку кодової бази та мають низький рівень зв’язку між компонентами моделі. Це робить її ефективною для пошуку помилок у коді та робить, за необхідності, інтеграцію нових компонентів легшою.

Практична перевірка працездатності моделі дозволяє дійти висновку, що усі завдання кваліфікаційної роботи виконано у повному обсязі, а запропонована модель є ефективною для використання у контексті тестування адаптивних вебзастосунків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Косінов Михайло, Мястковська Марина. Практична реалізація автоматизованих систем для здійснення поточного контролю здобувачів вищої освіти. Проблеми та інновації в математичній, цифровій, природничій і професійній освіті: збірник матеріалів XVIII-ї Міжнародної науково-практичної онлайн інтернет конференції, м. Кропивницький, 20 – 27 листопада 2024 року / Відп. ред. М. І. Садовий. Укладачі: М.І. Садовий, А.В. Бевз, О.М. Трифонова. Кропивницький: Інформаційний відділ ЦДУ ім. В. Винниченка, 2024. С. 136-137. URL: https://cusu.edu.ua/images/conferences/2024/Tezy_18_konf.pdf (дата звернення: 25.10.2025).
2. Косінов Михайло, Мястковська Марина. Перспективи вивчення тестування програмного забезпечення у закладах вищої освіти з метою покращення якості освіти. Актуальні аспекти розвитку STEAM-освіти в умовах євроінтеграції: збірник матеріалів III Міжнародної науково-практичної інтернет-конференції (м. Кропивницький, 24 квітня 2025 року). Кропивницький : ДонДУВС, 2025. С. 586-588. URL: https://dnuvs.ukr.education/wp-content/uploads/2025/06/zbirnyk_tez_konferencziya_steam_24_04_2025.pdf (дата звернення: 26.10.2025).
3. Косінов М., Мястковська М. Аспекти проєктування фреймворку автоматизованого тестування, які дозволяють покращити якість тестування. Проблеми та інновації в математичній, цифровій, природничій і професійній освіті: збірник матеріалів XIX-ї Міжнародної науково-практичної онлайн-інтернет конференції, м. Кропивницький, 20 – 29 травня 2025 року / Відп. ред. М.І. Садовий. Укладачі: М.І. Садовий, Д.В. Решетнікова, О.М. Трифонова. Кропивницький: Інформаційний відділ ЦДУ ім. В. Винниченка, 2025. С.117-119. URL: https://www.ldftpo.kr.ua/wp-content/uploads/2025/10/Tezu_XIX_konf.pdf (дата звернення: 27.10.2025)

4. Михайло КОСІНОВ. Особливості написання автоматизованих тестів за допомогою Playwright. Збірник наукових праць студентів та магістрантів Кам'янець-Подільського національного університету імені Івана Огієнка. [Електронний ресурс]. Кам’янець-Подільський: Кам’янець Подільський національний університет імені Івана Огієнка, 2025. Вип. 19. С. 137-139. URL: http://elar.kpnu.edu.ua/xmlui/handle/123456789/9147 (дата звернення: 01.11.2025)
5. Косінов М.С. Сучасні фреймворки для автоматизації тестування: інструменти для різних платформ. Вісник Кам’янець-Подільського національного університету імені Івана Огієнка. Фізико-математичні науки. Випуск 17. Кам’янець-Подільський : Кам’янець-Подільський національний університет імені Івана Огієнка, 2024. С. 63-65. URL: https://fizmat.kpnu.edu.ua/wp-content/uploads/2024/12/visnyk-17-2024-17-12-2024.pdf (дата звернення: 02.11.2025).
6. Косінов М.С. Особливості інтеграції CI/CD процесів для автоматизованого тестування вебзастосунків на базі Github Actions. Вісник Кам’янець-Подільського національного університету імені Івана Огієнка. Фізико-математичні науки. Випуск 18. Кам’янець-Подільський : Кам’янець-Подільський національний університет імені Івана Огієнка, 2025. С. 63-65.
7. Calculating Test Automation ROI: A Guide. URL: https://www.browserstack.com/guide/calculate-test-automation-roi (Дата звернення: 09.10.2025)
8. Sebastian Balsam, Deepti Mishra. Web application testing — Challenges and opportunities. Journal of Systems and Software. Volume 219, January 2025, 112186. DOI:10.1016/j.jss.2024.112186. URL: https://www.researchgate.net/publication/383209867_Web_application_testing-Challenges_and_opportunities (Дата звернення: 10.10.2025)
9. Принципи тестування. URL: https://qalight.ua/baza-znaniy/printsipi-testuvannya/ (Дата звернення: 11.10.2025)

10. Shoaib Farooq, Rabia Tehseen, Uzma Omer, Shamyla Riaz, Saba Tahir Software Testing Education: A Systematic Literature Review December 2021VFAST Transactions on Software Engineering 9(4):109-125 DOI: 10.21015/vtse.v9i4.953 URL: https://www.researchgate.net/publication/377405313_Software_Testing_Education_A_Systematic_Literature_Review (Дата звернення: 12.10.2025)
11. Maurizio Leotta, Andrea Stocco, Filippo Ricca, Paolo Tonella. ROBULA+: An Algorithm for Generating Robust XPath Locators for Web Testing. DOI: https://doi.org/10.1002/smr.1771 URL: https://tsigalko18.github.io/assets/pdf/2016-Leotta-JSEP.pdf (Дата звернення: 13.10.2025)
12. Andrea Stocco, Maurizio Leotta, Filippo Ricca, Paolo Tonella. APOGEN: Automatic Page Object Generator for Web Testing. DOI: http://dx.doi.org/10.1007/s11219-016-9331-9 URL: https://sepl.dibris.unige.it/publications/2016-stocco-SQJ.pdf (Дата звернення: 14.10.2025)
13. Test Vista Docs URL: https://docs.agentvista.ai/testvista/introduction (Дата звернення: 12.10.2025)
14. Що таке Low-Code/No-Code? URL: https://payproglobal.com/uk/%D0%B2%D1%96%D0%B4%D0%BF%D0%BE%D0%B2%D1%96%D0%B4%D1%96/%D1%89%D0%BE-%D1%82%D0%B0%D0%BA%D0%B5-low-code-no-code/ (Дата звернення: 12.10.2025)
15. Ткачик Д.А. Розробка через поведінку, як одна з методологій розробки програмного забезпечення ВНТУ. URL: https://conferences.vntu.edu.ua/index.php/mn/mn2020/paper/viewFile/10541/8834 (Дата звернення: 13.10.2025)
16. Behavior-Driven Development (BDD): що це таке? URL: https://foxminded.ua/bdd-shcho-tse/ (Дата звернення: 13.10.2025)
17. What is Gherkin? URL: https://www.browserstack.com/guide/what-is-gherkin

(Дата звернення: 14.10.2025)

18. Keyword driven testing in software development. URL: https://www.tricentis.com/learn/keyword-driven-testing (Дата звернення: 15.10.2025)
19. Understanding Page Object Model (POM) in Selenium. URL: https://www.qatouch.com/blog/page-object-model-in-selenium/ (Дата звернення: 16.10.2025)
20. Що таке Selenium? URL: https://qagroup.com.ua/publications/shcho-take-selenium/ (Дата звернення: 16.10.2025)
21. A Brief History of the Selenium Testing Framework. URL: https://www.testingmind.com/a-brief-history-of-the-selenium-testing-framework/ (Дата звернення: 17.10.2025)
22. Mastering Selenium IDE: A Comprehensive Guide for Test Automation. URL: https://www.lambdatest.com/learning-hub/selenium-ide (Дата звернення: 18.10.2025)
23. Selenium WebDriver Architecture Explained. URL: https://www.lambdatest.com/blog/selenium-webdriver-architecture/#:~:text=architecture%20consists%20of%20four%20major,components (Дата звернення: 19.10.2025)
24. Testing is the Key to Continuous Innovation. The Story of Cypress.io URL: https://www.cypress.io/about-us/our-story#:~:text=The%20application%20that%20became%20Cypress,team%20could%20use%2C%E2%80%9D%20he%20said (Дата звернення: 20.10.2025)
25. Cypress Docs. Trade-offs. URL: https://docs.cypress.io/app/references/trade-offs#:~:text=,our%20Cross%20Origin%20Testing%20Guide (Дата звернення: 21.10.2025)
26. Cypress Docs. Migrating from Selenium to Cypress. URL: https://docs.cypress.io/app/guides/migration/selenium-to-cypress#:~:text=2 (Дата звернення: 22.10.2025)
27. How to run Cypress Tests on Mobile Browsers. URL:

https://www.browserstack.com/guide/how-to-run-cypress-tests-on-mobile-browsers (Дата звернення: 23.10.2025)

28. Stages of Automation Testing Life Cycle. URL: https://www.geeksforgeeks.org/software-testing/stages-of-automation-testing-life-cycle/ (Дата звернення: 24.10.2025)
29. Certified Tester Advanced Level. Test Automation Engineering. Syllabus Version 2.0. URL: https://istqb.org/wp-content/uploads/2024/11/ISTQB_CTAL-TAE_Syllabus_v2.0.pdf (Дата звернення: 25.10.2025)
30. Defect Cost Model to improve software productivity URL: https://medium.com/@SWQuality3/what-is-orthogonal-defect-classification-odc-by-vivek-vasudeva-2f5494622685 (Дата звернення: 26.10.2025)
31. Axios Docs URL: https://axios-http.com/uk/docs/intro (Дата звернення: 27.10.2025)
32. Playwright Docs. Test Isolation URL: https://playwright.dev/docs/browser-contexts (Дата звернення: 28.11.2025)
33. Архітектурна модель реалізації Page object model. URL: https://testomat.io/blog/page-object-model-pattern-javascript-with-playwright/ (Дата звернення: 29.10.2025)
34. Diagrams.net (draw.io). Online diagram editor. URL: https://app.diagrams.net/ (Дата звернення: 30.10.2025).

ДОДАТКИ

Додаток А. Існуючі підходи до автоматизації тестування

Тестування за допомогою AI-інструментів

Огляд існуючих рішень для автоматизації тестування показує, що є значна розбіжність між академічними дослідженнями та промисловими потребами [10]. Значна кількість досліджень не була розглянута у реальних практичних умовах реалізації проєктів, що ставить під сумнів практичну цінність розглянутих підходів. В умовах сучасного IT-бізнесу важливими є короткі терміни розробки і проведення тестування нових систем, а також, впровадження доопрацювань і нових функціональних можливостей у вже існуючі проєкти.

Прикладами існуючих академічних інструментів, які допомагають у автоматизованому тестуванні є:

1. “Robula+” – реалізовує алгоритм для генерації надійних локаторів веб-елементів, що зменшує крихкість тестів [11].

2. “Apogen” – автоматично генерує об’єкти коду веб-сторінок, вирішуючи проблему підтримки кодової бази фреймворку автоматизації. Обмежений реалізацією лише для мови програмування Java [12].

3. “Vista” – забезпечує підтримку тестових сценаріїв відповідно до UI-інтерфейсу застосунків, використовуючи алгоритми комп’ютерного зору для визначення і виправлення знайдених невідповідностей [13].

Дані рішення демонструють вирішення точкових проблем у сучасній автоматизації тестування, але мають перелік недоліків, які роблять їх “слабким” конкурентом для інструментів, що надають комплексні рішення для тестуванні складних проєктів. Серед недоліків академічних рішень допустимо визначити:

- відсутність комплексного рішення, яке дозволяє здійснювати тестування усього програмного продукту;

- підтримка однієї або кількох мов програмування, що робить

неможливим використання даних інструментів існуючого широкого переліку проєктів, реалізованих за допомогою різних технологій;

- відсутність апробації та підтвердження ефективності у комерційних проєктах, що призводить до зростання ризиків з точки зору доцільності використання у бізнесі.

Найновішою тенденцією у автоматизації тестування є використання AI-рішень, подібних до “Vista”, “ACCELQ Autopilot”, “Mabl”. Особливість даних інструментів полягає в тому, що вони використовують “no-code” та “low-code” рішення. Це означає, що автоматизовані тести створюються описовим методом, без потреби написання коду тестувальниками. Це призводить до зменшення порогу входу в автоматизацію нетехнічним командам, дозволяє створювати рішення, які здатні генерувати і підтримувати певний набір тестових сценаріїв без безпосередньої участі людини [14]. Проте, до характерних недоліків AI рішень відносять наступні чинники:

- висока вартість впровадження у проєкт;

- складність використання у специфічних доменах, таких як електронний документообіг, медицина та, домени, де важливу роль відіграють відповідність нормативним документам та забезпечення захисту і прозорості використання персональних даних користувачів;

- недостатня кількість відомостей про ефективність використання повноцінних AI рішень у комерційних проєктах.

Page Object Model

Автоматизоване тестування вебзастосунків передбачає від тестувальників наявності експертизи у певних аспектах реалізації систем [2]. У період коли на просторах інтернету більшість вебзастосунків представляли собою набір статичних веб-сторінок, зародився один з перших патернів автоматизації “Page object model”.

Page object model – шаблон проєктування автоматизованих тестів, у якому кожна з веб-сторінок представлена відповідним класом, який описує сторінку системи. Даний підхід дозволяє зробити код з тестами читабельним,

підтримуваним та масштабованим, оскільки реалізація опису системи за допомогою класів дозволяє отримати більше відомостей про структуру системи, та як наслідок, дозволяє реалізувати повторне використання коду у різних тестах. Розподілення логіки надає тестувальникам можливість працювати паралельно над написанням тестових сценаріїв, без поглиблення в технічну логіку опису веб-сторінок.

Структурно POM реалізується шляхом розділення логіки автоматизованого фреймворку на дві основні частини – тестові сценарії та кодова база, яка містить класи сторінок веб-системи. Відповідні класи можуть бути розширеними, шляхом використання наслідування класів. Оскільки додаток представлений набором сторінок, то в основі POM лежить базовий класс BasePage, який реалізує методи доступні для всіх дочірніх класів. Прикладами таких методів є таймери для очікування певних елементів інтерфейсу, або подій чи модальних вікон, отримання cookies, токенів авторизації, даних, які можуть зберігатися у LocalStorage браузеру.

Усі класи сторінок наслідуються від BasePage та можуть бути представлені класами, які можуть бути декомпозовані на додаткові класи, що описують повторювані елементи веб-сторінок або певні секції. Такий підхід дозволяє отримати наочне уявлення про взаємодію компонентів системи та може принести значну користь в умовах відсутності або недостатньої кількості документації системи. Модель реалізації архітектури POM представлена на рис. А.1.

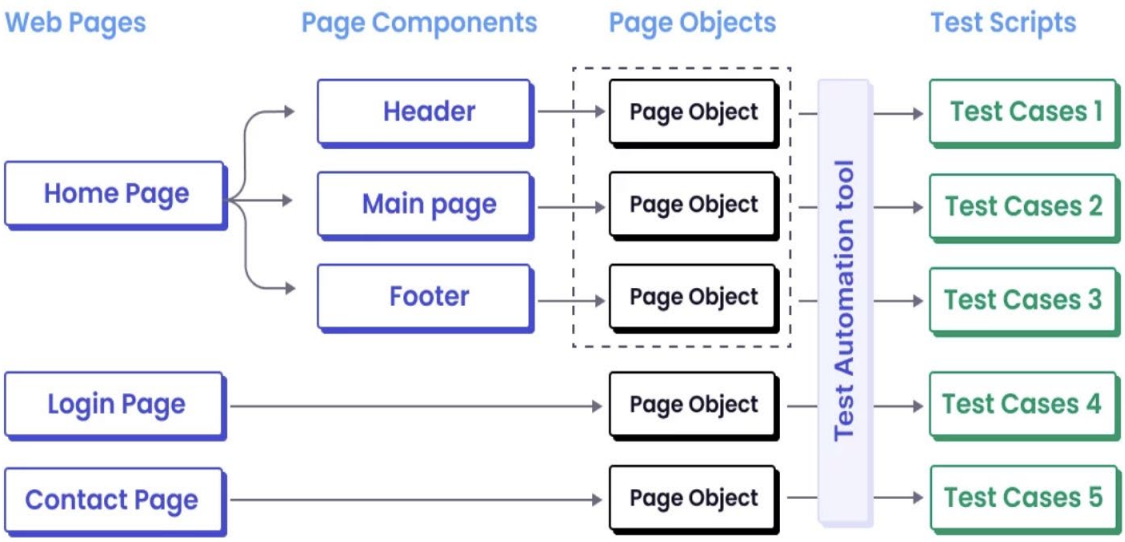


Рис. А.1. Архітектурна модель реалізації Page object model. Джерело: Page Object Model Pattern: JavaScript With Playwright. Режим доступу: https://testomat.io/blog/page-object-model-pattern-javascript-with-playwright/

Елементи веб-сторінок представляються у вигляді об’єктів та методів, що доступні для взаємодії з відповідними елементами. Як наслідок, характерною особливістю та однією з основних переваг POM є використання локаторів (унікальних ідентифікаторів) елементів інтерфейсу всередині класів веб-сторінок. Оскільки, взаємодія з інтерфейсом відбувається за допомогою описаних методів класу, то при зміні ідентифікаторів, необхідною буде лише оновлення значення відповідного локатору. Такий підхід не створює проблем для оновлення набору тестів, бо тестові сценарії є ізольованими від технічної реалізації фреймворку, а дозволяє зосередитись на бізнес-логіці, з метою забезпечення вищої якості програмного продукту.

Behavior-Driven Development

Розробка керована поведінкою (англ. “Behavior-Driven Development”) – це процес розробки програмного забезпечення, який виник з Test Driven Development (TDD). В основі даного підходу лежить опис і документування роботи системи у вигляді користувальницьких сценаріїв використання системи [15]. Це дозволяє стейкхолдерам, розробникам, тестувальникам та усім іншим учасникам реалізації проєкту глибше зосередитися на очікуваних

результатах роботи системи. Основним призначенням BDD є покращення наступних елементів розробки програмного продукту:

- забезпечення належного рівня комунікації між командами;
- зосередження на глибокому розумінні вимог – дозволяє зменшити кількість конфліктних ситуацій між командами, оскільки всі виконавці мають однозначне бачення очікуваної поведінки роботи системи;
- раннє виявлення помилок – підхід опису поведінки системи дозволяє виявити більшу кількість проблем на етапі планування, ще до початку практичної реалізації проєкту;
- підтримка та масштабування – оскільки функціонал системи має зрозумілу і передбачувану поведінку, то впровадження нових функціональних модулів надає командам краще розуміння про зміни, які є необхідними для реалізації;
- стабільність автоматизації тестування – зважаючи на те, що тестові сценарії проєктуються на основі поведінки системи, тестувальники з більшою ймовірністю можуть бути впевнені у відсутності небажаних або неочікуваних побічних ефектів, які виникли внаслідок внесених змін у систему [16].

Для впровадження BDD підходу у тестування, як правило, використовують сценарії описані природною для людини мовою за допомогою спеціального синтаксису Gherkin. Даний синтаксис описує кожен тестовий сценарій за допомогою набору ключових слів, серед яких “Given”, “When” та “Then” [17].

Використання даного синтаксису надає структурований опис роботи системи, для певного користувальницького сценарію, який використовується як документація аналітиками, розробниками та тестувальниками під час розробки.

В основі кожного ключового слова певний стан системи, що тестується, а саме:

1. “Given” – описує початковий стан системи, перед початком виконання тестування.

2. “When” – включає в себе повний перелік дій, за допомогою яких користувач взаємодіє з системою.

3. “Then” – під даним ключовим словом виконується очікуваний опис системи, після того, як система в результаті виконання певних дій перейшла з початкового стану в очікуваний. Цей крок є фіналізуючим, та дозволяє здійснити верифікацію певних вимог.

Відповідно до наведених особливостей реалізації BDD підходу спостерігається послідовний зв’язок, який представляє собою життєвий цикл представлений на рисунку А.2.

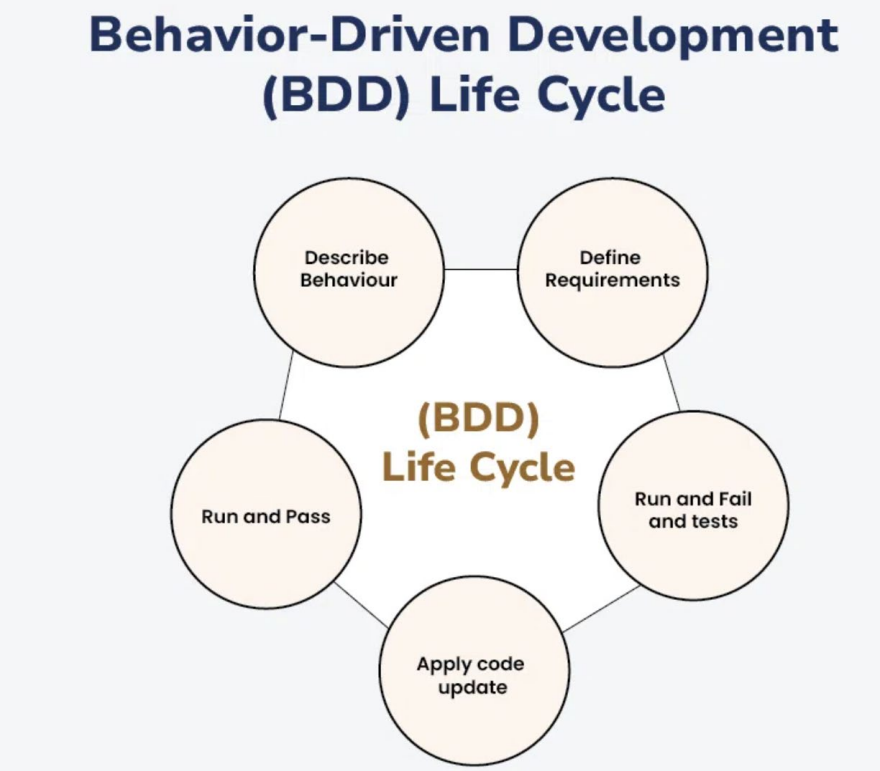


Рис. А.2. Життєвий цикл BDD. Джерело: What is Behavior-Driven Development (BDD)? Режим доступу: https://www.geeksforgeeks.org/software-engineering/behavioral-driven-development-bdd-in-software-engineering/

Першим етапом циклу є опис поведінки системи, яка є очікуваною після завершення розробки. Наступним кроком є визначення вимог до системи, які можуть бути описані за допомогою Gherkin-синтаксису, який є доступним для технічних і нетехнічних фахівців, які залучені до розробки. Після визначення вимог тестувальники приступають до розробки автоматизованих тестів, а розробники до написання коду. Під час цього тестові сценарії можуть бути в

статусі “Fail”, через те, що частина функціоналу ще не є реалізованою, або ж через наявні дефекти. Четвертим етапом життєвого циклу BDD підходу є внесення оновлень та доопрацювань дефектів, виявлених на попередньому етапі тестування. Життєвий цикл завершується виконанням повторного тестування, яке задовольняє вимогам замовників та користувачів.

Спираючись на досліджені особливості BDD допустимо зробити висновки, які підкреслюють переваги та недоліки даного підходу.

Переваги використання для тестування:

- покращення комунікації та розуміння вимог до продукту, завдяки використанню синтаксису Gherkin;
- фокус на розробці системи через поведінку, що дозволяє чітко визначити пріоритети і важливість функціонування окремих компонентів системи;
- раннє виявлення дефектів – дозволяє виявляти певні дефекти, ще до початку активної розробки;
- використання природної мови для опису системи.

Натомість, у противагу переліченим позитивним аспектам BDD підходу можна відзначити наступні недоліки:

- складність використання у розробці програмних продуктів за методологією Agile, оскільки ітераційні зміни до вимог створюють нові виклики для розробників та тестувальників, зокрема значною мірою збільшують час на підтримку і оновлення документації та виконання тестування;
- значні початкові витрати на отримання навичок роботи з синтаксисом Gherkin та початковими налаштуваннями тестового середовища;
- зосередженість на функціональному тестуванні – призводить до того, що утворюються “пробіли” у тестуванні нефункціональних вимог (UI/UX, тестування безпеки, продуктивності та інші).

Таким чином, використання BDD підходу є ефективним у випадку, коли є чітко сформовані вимоги та команди мають належний рівень навичок роботи

з Gherkin-сценаріями. За відсутності наявності даних критеріїв використання BDD підходу з більшою ймовірністю може створити більше нових викликів, ніж переваг.

Keyword driven testing

Тестування на основі ключових слів – один з підходів тестування, де тест-кейси будуються на основі ключових слів, які описують виконання певних дій користувача [18]. Даний підхід реалізується шляхом оформлення переліку ключових слів (частіше за допомогою таблиці або окремого файлу). Кожне відповідне слово відповідає певній функції, яка описує кодову реалізацію певних дій. Це дозволяє абстрагуватися від технічної реалізації та зосередитись на написанні автоматизованих скриптів та цілями тестування. Обов'язковою вимогою для виконання тестових сценаріїв є використання двигуна, який призначений для співставлення і виконання функцій які співставлені з ключовими словами.

Підхід тестування на основі ключових слів дозволяє зробити тестові скрипти зрозумілими, для фахівців з різною технічною експертизою, оскільки тестувальник, зазвичай, не виконує завдань для оновлення і реалізації даних слів. Зважаючи на це, поріг входу в автоматизацію зменшується для молодих спеціалістів та мануальних тестувальників, які наразі не володіють глибокими навичками у програмуванні.

Фреймворк автоматизації тестування за допомогою KDT включає в себе наступний перелік основних елементів:

1. Бібліотека ключових слів – містить перелік усіх ключових слів, які використовуються в автоматизації тестування.
2. Common libraries – набір бібліотек, які містять в собі перелік функцій, які реалізують послідовності дій, які асоціюються з певним ключовим словом.
3. Тестові дані – набір даних, які використовуються для тестування з різними наборами даних.
4. Тест-скрипти – набір автоматизованих тестових сценаріїв, які написані за допомогою використання ключових слів.

5. Двигун виконання – призначений для запуску і керування виконання тестових скриптів за відповідними кроками.

Даний підхід надає можливість локалізації проблем, які можуть виникати під час тестування. Схематична модель архітектури KDT фреймворку представлена на рис. А.3.

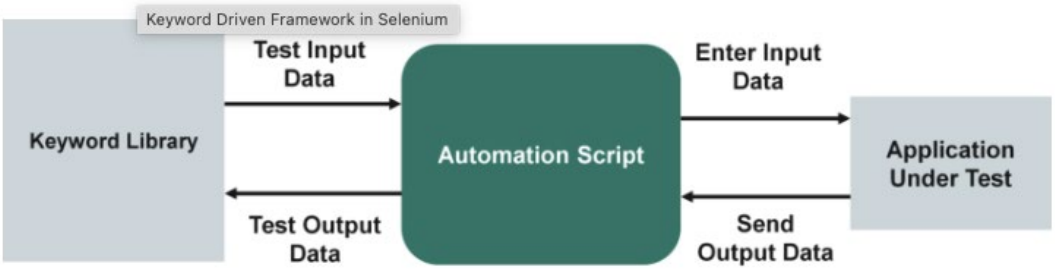


Рис. А.3. Модель архітектури KDT фреймворку. Джерело: Keyword Driven Framework for Selenium. Режим доступу: https://www.browserstack.com/guide/keyword-driven-framework-in-selenium

До переваг KDT-патерну в автоматизації допустимо віднести:

- зниження порогу входу в автоматизацію для тестувальників;
- спрощення підтримки тестових сценаріїв, оскільки оновлень потребують функції, які викликаються за допомогою ключових слів;
- структурований і чіткий розподіл логіки;
- висока читабельність тестів.

Попри всі переваги, даний підхід потребує значних часових і вартісних витрат на початкове налаштування та проєктування фреймворку. Також, використання для короткотривалих і часто змінюваних проєктів створює додаткові перепони для виконання тестування, оскільки реалізація функцій, що реалізують послідовності дій описаних ключовими словами вимагатимуть частого оновлення і підтримки, що суперечить основним цілям KDT підходу.

Додаток Б. Існуючі фреймворки автоматизації вебзастосунків Selenium

Під час зародження онлайн сервісів для збереження інформації, її обліку, та продажу товарів вебзастосунки перестали бути статичними сторінками, які відображали інформацію. Виконання одних і тих самих сценаріїв у браузері було задачею, яка потребувала значних часових витрат та допускала високу ймовірність помилок під час тестування. Саме для вирішення цієї проблеми у 2004-му році Джейсоном Хаггінсом було випущено перший інструмент автоматизації тестування вебзастосунків під назвою Selenium [21].

В межах екосистеми селеніума існує перелік з чотирьох програмних продуктів: Selenium WebDriver, Selenium RC, Selenium IDE, Selenium Server+Selenium Grid. Наразі, усі ці програмні продукти часто іменують однойменно – Selenium. Проте, кожен з цих інструментів має своє призначення та залишив свій вплив на сучасну автоматизацію вебзастосунків.

Selenium IDE – середовище розробки, яке дозволяє запам’ятовувати дії користувача у браузері та повторно виконувати їх у вигляді автоматизованих тест-кейсів. Основними недоліками використання даного інструменту є повільна швидкість роботи та низький рівень підтримки та оновлення наявних автоматизованих тестів [22].

Selenium Remote Control – засіб віддаленого керування браузеру за допомогою мов програмування. Даний інструмент був створений, як перший інструмент автоматизації веб-тестування та об’єднаний з Selenium WebDriver у версії 2.

Selenium Webdriver – основний продукт Selenium. Він представляє собою набір клієнтських бібліотек, JSON-протокол, драйвери браузеру та справжні браузери. Набір клієнтських бібліотек робить можливою використання фреймворку з використанням різних мов програмування, таких як Javascript, C#, Python, Ruby, Java. Вони надають мовні прив’язки, які містять набір функцій та методів, завдяки яким тестувальники можуть взаємодіяти з браузером.

JSON Wire протокол використовується для перетворення команд до браузера в HTTP-пакети та їх відправки до драйверу браузеру. Даний протокол робить можливою взаємодію клієнтських бібліотек з драйвером браузера, та є допоміжним шаром, який забезпечує однакову поведінку браузера коли використовуються різні мови програмування. Даний протокол був замінений на протокол W3C. Необхідність даного переходу полягала у тому, що JSON Wire розумів лише протоколи, а виконання тестів було повільним та призводило до нестабільності тестів. Саме тому W3C протокол став ефективною заміною старого протоколу, оскільки не потребує перетворення команд до драйверу браузера в HTTP-запити, оскільки тестові дії напряму передаються браузеру драйвера [23].

Особливими рисами Selenium WebDriver є керування браузером з рівня операційної системи, його використання можливе з різними мовами програмування та реалізація додаткової гнучкості написання автоматизованих тестів з використанням циклів та умовних конструкцій відповідних мов програмування. Додатково, можливість виконання тестів у headless режимі дозволяє економити час на виконання тестів, оскільки час очікування на відображення веб-сторінок у браузері не відбувається.

Сучасна архітектура фреймворку Selenium версії 4 представлено на рисунку Б.1.

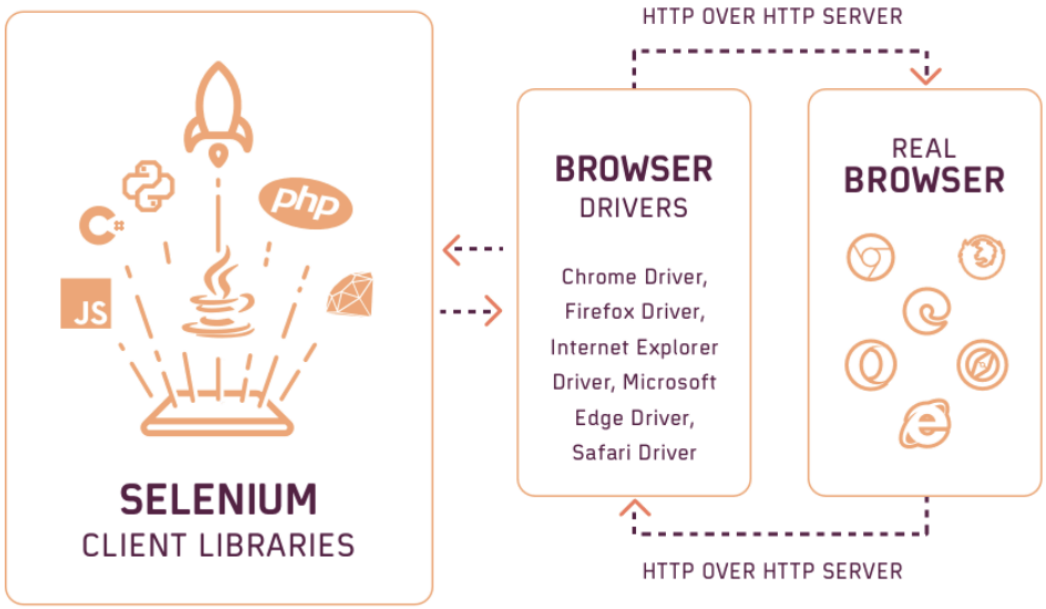


Рис. Б.1. Архітектура фреймворку Selenium

Джерело: Selenium WebDriver Architecture Explained. Режим доступу: https://www.lambdatest.com/blog/selenium-webdriver-architecture/#:~:text=With%20the%20introduction%20of%20Selenium,0

Незважаючи на низку переваг, серед яких кросбраузерність, підтримка багатьох мов програмування, емуляція різних типів пристроїв фреймворк Selenium має низку недоліків:

- відсутність вбудованої звітності про результати тестування;
- складність підтримки останніх версій браузера, через необхідність очікувати оновлень WebDriver API;
- багатопотокове виконання тестів не є вбудованим у Selenium за замовчуванням;
- складність підтримки тестів з асинхронністю у сучасних SPA-застосунках.

Архітектура фреймворку автоматизації Selenium стала основою для автоматизації веб-застосунків з підтримкою адаптивності та кросбраузерності. Підтримка різних типів браузерів та постійні оновлення роблять даний фреймворк ефективним інструментом автоматизації, проте відсутність вбудованої звітності, паралельного виконання тестів ускладнюють його

використання у сучасних веб-проєктах, які мають складну структуру та великі набори регресійних кейсів.

Cypress

Використання існуючого фреймворку Selenium впродовж років дозволило тестувальникам зіштовхнутись з низкою проблем та обмежень, які ускладнювали процеси автоматизації тестування. У 2014 році Браян Манн розпочав розробку нового фреймворку автоматизації для вирішення поточних проблем його команди. У 2018 році відбувся офіційний реліз у вигляді релізу комерційного проєкту під ліцензією MIT, після чого популярність даного інструменти почала швидко зростати та привернула увагу технологічних гігантів, таких як Slack, Disney, NBA, Netflix та інших [24].

Характерна відмінність фреймворку Cypress від його попередника Selenium лежить у його архітектурі. Cypress здійснює запуск застосунку, що тестується та тестів безпосередньо у браузері. Даний фреймворк, замість відправки запитів до браузеру ззовні, як це відбувається у Selenium, створює процес Node.js, який дозволяє виконувати тести в одному контексті з роботою додатку. Це дозволило зробити результати тестування більш передбачуваними та стабільними, зменшило час на виконання команд, оскільки команди напряму надсилаються до браузеру. Додатковою, особливою відмінністю фреймворку Cypress є наявність вбудованого HTTP-проксі, який дозволяє перехоплювати мережеві запити та відповіді на них. Таким чином, виникає можливість створення “заглушок” для написання компонентних та інтеграційних тестів. Прямий доступ до структури веб-сторінки, мережі та до файлової системи дозволяє здійснювати “ін’єкції коду”, які дозволяють гнучко керувати процесом виконання тесту на кожному його кроці.

Закладені підходи в архітектуру фреймворку Cypress створюють низку обмежень на його використання, зокрема, лише з мовою програмування JavaScript. Проте, можливість створення “заглушок” та контролю і керування кожним кроком тестового сценарію робить Cypress популярним інструментом для автоматизації тестування фронтенду.

Додатковим обмеженням фреймворку Cypress, яке не є технічною вадою, а свідомим рішенням розробників – відсутність підтримки тестів, які потребують відкриття кількох вкладок чи вікон браузеру. Це пов’язано з закладеною у фреймворк філософією “Один браузер – один контекст”, яка спрямована на досягнення детермінованості і простоти тестових сценаріїв. Оскільки тести є короткими та виконуються в межах однієї сторінки, то час їх виконання, підтримки та стабілізації зменшується [25].

Використання фреймворку Cypress пропонує тестувальникам широкий набір інструментів, які дозволяють зосередитись на тестуванні самого вебзастосунку, а не автоматизиції взаємодії з браузером. Серед особливостей і переваг фреймворку допустимо визначити:

- автоматичні очікування завантаження елементів веб-сторінок;
- простота початкового налаштування – для написання першого автоматизованого скрипта відсутня необхідність проведення додаткових налаштувань;
- наявність вбудованої звітності;
- можливість покрокового відстеження виконання тесту (дебаггінг);
- відсутність мережевих затримок – сприяє надійності і швидшому виконанню тестів;
- можливість робити скріншоти та записувати відео виконання тестів;
- активна підтримка спільнотою та широкий набір бібліотек і плагінів, які дозволяють обходити наявні обмеження фреймворку [26].

Широкий набір функціональності, простота налаштування і початку написання автоматизованих тестів робить фреймворк Cypress ефективним інструментом автоматизації тестування, який дозволяє збільшити покриття автоматизацією завдяки можливості написання не лише end-to-end тестів, але й компонентних та інтеграційних. Даного результату розробникам вдалось досягти завдяки альтернативній архітектурі фреймворку, яка представлена на рисунку Б.2.

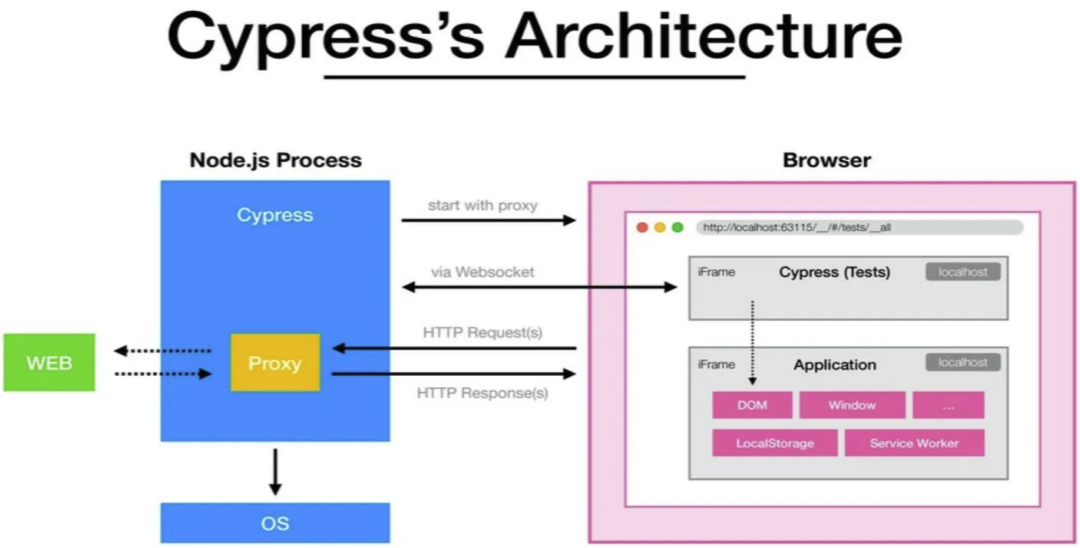


Рис. Б.2. Архітектура фреймворку Cypress

Джерело: Cypress Architecture: A Detailed Exploration with Real-Time Examples. Режим доступу: https://medium.com/@akssingh002/cypress-architecture-a-detailed-exploration-with-real-time-examples-d5cbf02b1030

Архітектура фреймворку складається з наступних компонентів: серверний процес Node.js; test-runner; браузер; мережевий проксі-сервер; доступ до файлової системи; API та плагіни.

Основою для роботи фреймворку є взаємодія процесу Node.js, test-runner`у та браузеру. За допомогою системного процесу, який є основою архітектури системи, фреймворк обробляє зв'язок між браузером та test runner`ом, керує читанням, записом файлів та взаємодіє з операційною системою для виконання мережевих операцій. Test runner забезпечує інтерфейс виконання автоматизованих тестових сценаріїв та відповідає за запуск тестового коду, надання інструментів для відлагодження і стабілізації скриптів.

Фреймворк Cypress є гарним і ефективним рішенням для побудови автоматизації тестування для десктопних веб-застосунків, оскільки має широкий набір функціональності та дозволяє покрити більшість сценаріїв взаємодії користувача з браузером, робить процес автоматизації тестування передбачуваним та найдійним, у порівнянні з Selenium.

Вагомим недоліком використання фреймворку Cypress для тестування

адаптивних веб-застосунків є обмежена підтримка кросбраузерності та відсутність справжньої емуляції мобільних пристроїв. Даний фреймворк працює лише з десктопними версіями браузеру та не має підтримки жестових подій, таких як tap, swipe, tap-and-hold, які є властивими для мобільних пристроїв [27]. Таким чином, відсутність належної підтримки даної функціональності є вагомим недоліком для автоматизації тестування адаптивних веб-застосунків, оскільки це призводить до зменшення кількості сценаріїв, які можуть бути автоматизовані та потенційно призводить до збільшення кількості дефектів, які можуть бути не виявлені під час проведення циклу тестування.

Додаток В. Використання та тестування моделі

Створення набору автоматизованих тестів

Для тестування моделі і перевірки її працездатності необхідно спроєктувати та автоматизувати певний набір тестових сценаріїв. Для тестування моделі було обрано у якості об’єкту тестування демонстраційний веб-сайт “Automation Exercise” (URL: https://automationexercise.com/). Даний веб-сайт має адаптивний інтерфейс та призначений для отримання практичних навичок з автоматизації тестування, тому його можна безпечно використовувати для тестування і перевірки моделі [4].

Відповідно до завдань і цілей кваліфікаційної роботи магістерського рівня, модель повинна коректно працювати за наступних умов:

1. Зміни коду відбуваються лише всередині директорії scr та tests. У директорії scr описуються сторінки вебзастосунку та доступні дії з ними.
2. Набір реалізованих тестів успішно виконується на усіх доступних конфігураціях.

Для створення набору тестів призначених для перевірки моделі масштабованої архітектури фреймворку автоматизації адаптивних вебзастосунків описано наступні елементи системи:

- логін та реєстрація;
- домашня сторінка;
- сторінка товарів;
- сторінка товарів відсортована відповідно до обраних категорій;
- компонент навігаційного меню та модального вікна додавання товару у кошик.

На основі описаних сторінок описано і автоматизовано певний перелік тестових сценаріїв. Тестові сценарії згруповано за функціональністю, для перевірки яких вони призначені. Також, автоматизовані тести є повністю ізольованими. Для тестування моделі було реалізовано наступні тестові сценарії:

1. логін існуючого користувача;

2. логін з коректним паролем;
3. реєстрація нового користувача;
4. пошук товару за вказаним фільтром;
5. пошук користувача за допомогою поля пошуку;
6. перевірка наявності рекомендованих товарів на головній сторінці сайту;
7. додавання товару до кошику.

Вказаний набір тестів, включає в себе перевірки, пов’язані зі зміною відображення інтерфейсу на пристроях типу mobile та tablet.

Проведення тестування працездатності моделі

Виконання працездатності моделі виконано шляхом виконання створених 7 тестових сценаріїв для вказаних конфігурацій проєкту за замовчуванням. Для тестування моделі визначено наступні конфігурації:

1. desktop – браузери Chrome, Safari, Firefox;
2. tablet – браузери Chrome, Safari для пристроїв Galaxy S9+ та iPhone 12 Pro;
3. mobile – браузери Chrome, Safari для пристроїв Galaxy Tab S4 та iPad Pro 11.

Таким чином для тестування системи виконано 77 тестів, для різних конфігурацій. Принципи які закладені в моделі, повинні забезпечувати повноцінну здатність і гнучкість повторного використання коду написаних тестів на окремому рівні моделі, а класи, що описують сторінки, повинні бути адаптованими під виконання коду для різних типів інтерфейсу.

Відповідно до визначених до моделі масштабованої архітектури фреймворку автоматизації тестування адаптивних вебзастосунків тести виконувались у 5 потоків та за результатами тестування було сформовано два звіти Allure та HTML. Звіт Allure, демонструє перелік конфігурацій, статус виконання тестів та їх кількість наведено на рисунку В.1.

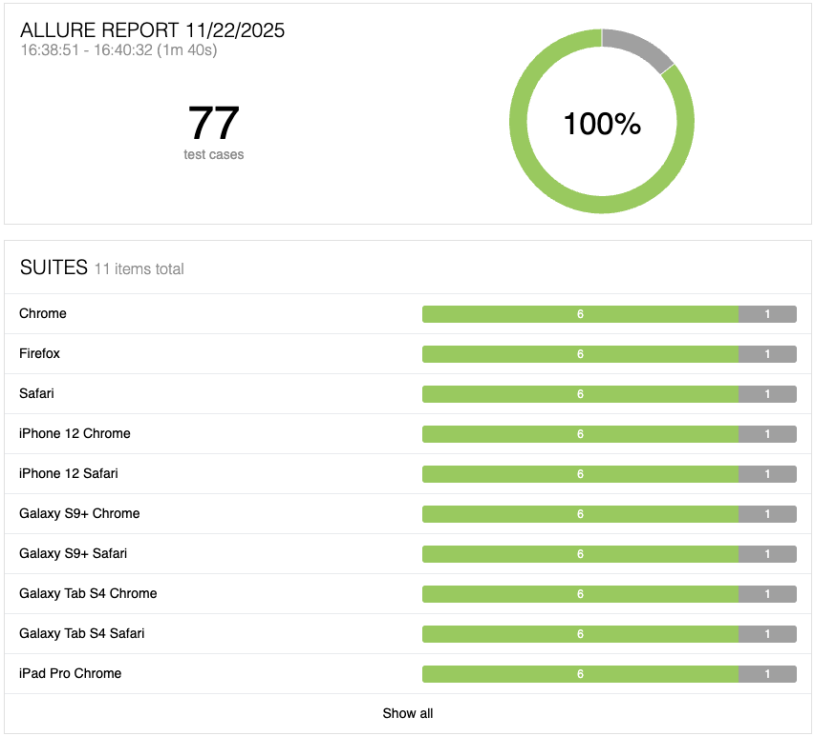


Рис. В.1. Allure звіт з результатами тестування моделі

Додатково, здійснено перевірку працездатності системи за допомогою використання Github Actions (рисунок В.2.).

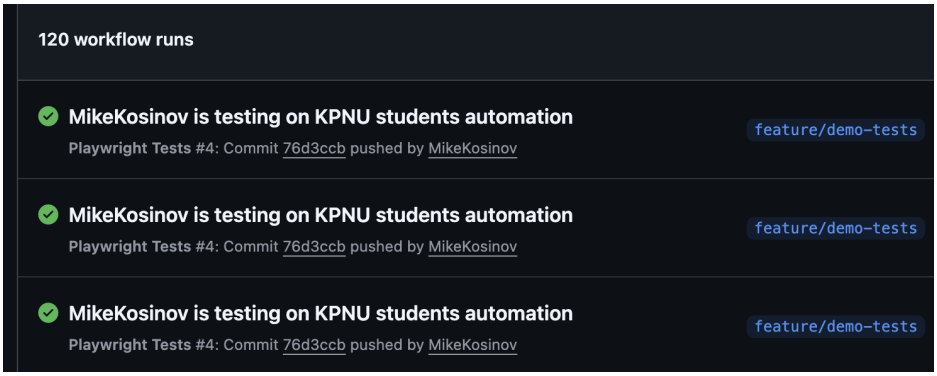


Рис. В.2. Звітність про успішне виконання тестування моделі у Github репозиторії

Аналіз результатів та перспективи використання у комерційних проєктах

Проведення аналізу результатів тестування моделі масштабованої архітектури фреймворку автоматизації тестування адаптивних вебзастосунків є важливим етапом, необхідним для підтвердження її працездатності та ефективності. Під час тестування моделі було виконано 7 тестових сценаріїв на 11-ти конфігураціях, визначених для різних типів пристроїв.

Успішне виконання тестів на різних конфігураціях дозволяє стверджувати, що модель має чітку рівневу архітектуру та відповідає вимогам написання ефективного коду, який є гнучким, прогнозованим, з мінімальним дублюванням коду та має чисту логіку з низьким рівнем зв’язності. Відповідно, дану модель можна вважати комплексною та масштабованою, оскільки розширення кодової бази відповідно до закладених у модель підходів матиме низький рівень впливу на вже реалізовану функціональність.

Використання запропонованої моделі у комерційних проєктах є важливим кроком до побудови ефективної стратегії тестування адаптивних вебзастосунків. Перелік визначених конфігурацій є змінюваним і легкопідтримуваним, що дозволяє легко адаптувати фреймворк під потреби тестування окремо визначеного продукту. Дана модель реалізовує рекомендовані підходи та практики, що дозволяє економити час на їх вивчення та практичну реалізацію з самого початку.

Отже, використання моделі несе пряму фінансову вигоду у вигляді скорочення витрат на початкову реалізацію, дозволяє уникнути проведення передчасного рефакторингу та надає достовірну, прозору і вичерпну інформацію про результати виконання тестування. Це свідчить про те, що модель очікувано буде користуватися попитом на комерційному ринку, як ефективний інструмент автоматизації адаптивних вебзастосунків.

Перспективи вдосконалення створеної моделі

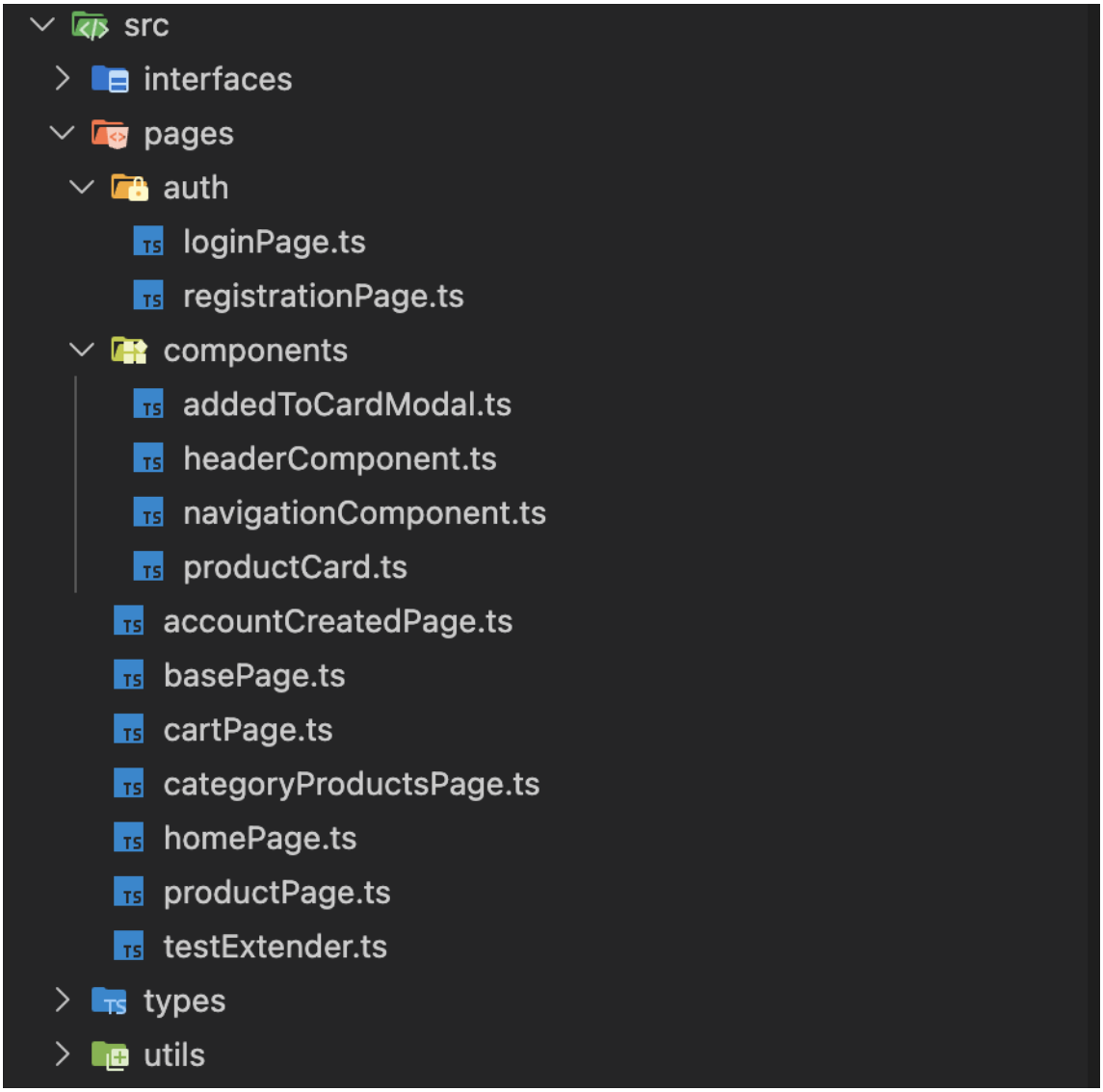
Велика кількість наявних адаптивних вебзастосунків та їх належність до різних доменних областей роблять модель вразливою, оскільки в ній не закладено особливості для рішень, які мають доменну специфіку. Саме тому, головним вектором спрямованим на вдосконалення моделі є реалізація користувальницького інтерфейсу, який дозволяє визначати перелік необхідних компонентів моделі відповідно до домену проєкту. Додатково, створення єдиної і формалізованої інструкції, яка буде розкривати особливість реалізації окремих компонентів моделі, дозволить пришвидшити процес адаптації нових членів команди тестування.

Висновки

У даному розділі описано процес формування набору тестів призначеного для тестування моделі масштабованої архітектури фреймворку автоматизації тестування адаптивних вебзастосунків. Проаналізовано результати тестування моделі та визначено, що модель відповідає визначеним вимогам, є масштабованою та ефективною. Розкрито та обґрунтовано особливості і переваги використання моделі у комерційних проєктах. Запропонована модель має перспективи отримати високий попит у комерційному просторі через реалізовану підтримку адаптивності, модульну структуру та відповідає практикам написання чистого коду.

Визначено перспективи вдосконалення функціональних можливостей моделі шляхом реалізації формування її структури відповідно до доменної специфіки адаптивних вебсистем. Також, важливим елементом для вдосконалення моделі буде створення детальної документації та структуризованого джерела інформації, яке дозволить зробити процес впровадження моделі простішим, а процес адаптації нових спеціалістів, які працюють над проєктом, простіше та коротше.

Додаток Г. Файлова структура застосунку з використанням POM



Додаток Д. Лістинг програмного коду тестів, призначених для тестування моделі

*Файл **tests/auth.spec.ts***

```
import { test } from '@testExtenter';
import { uiConst } from '@utils/constants/uiConst';
import { fakerDataGenerator } from '@utils/helpers/fakerGeneratedData';
import { LoginDataType } from 'src/types/userTypes';

test.describe(`Verify login page`, async () => {
 test.beforeEach(async ({ homePage }) => {
   await homePage.load();
 });

 test(`Login with existed user`, async ({ homePage, headerComponent, loginPage,
navigarationComponent }) => {
   const loginData: LoginDataType =
     process.env.email && process.env.password
       ? { email: process.env.email, password: process.env.password }
       : { email: 'test151@example.ocm', password: 'Test_12345' };
   await homePage.clickOnMenuItem(uiConst.navigationMenu.SignUpOrLogin);
   await loginPage.verifyPageURL('/login');
   await loginPage.fillAndSubmitLoginForm({
     email: loginData.email,
     password: loginData.password,
   });
   await headerComponent.verifyLoggedInMessage('Logged in as John joe');
   // test cleanup - logout user
   await navigarationComponent.clickOnMenuItem(uiConst.navigationMenu.Logout);
   await homePage.verifyPageURL('/login');
 });

 test('Login with invalid credentials', async ({ homePage, loginPage }) => {
   await homePage.clickOnMenuItem(uiConst.navigationMenu.SignUpOrLogin);
   await loginPage.verifyPageURL('/login');
   await loginPage.fillAndSubmitLoginForm({
     email: 'incorrect_mail@example.ocm',
     password: 'incorrect_password',
   });
   await loginPage.verifyInvalidCredentialsMessage();
 });
 test(`Register new user`, async ({ navigarationComponent, registrationPage,
headerComponent, loginPage, accountCreated }) => {
   const randomUser = fakerDataGenerator.generateNewUserData();
```

```
          await
navigarationComponent.clickOnMenuItem(uiConst.navigationMenu.SignUpOrLogin);
          await loginPage.verifyPageURL('/login');
          await loginPage.fillAndSubmitSignupForm({
            name: randomUser.name,
            email: randomUser.email,
          });
          await registrationPage.fillAndSubmitRegistrationForm(randomUser);
          await accountCreated.verifyAccountCreatedMessage();
          await accountCreated.clickOnContinueButton();
          await headerComponent.verifyLoggedInMessage(`Logged in as
${randomUser.firstname} ${randomUser.lastname}`);
          // test cleanup - delete created user
          await
navigarationComponent.clickOnMenuItem(uiConst.navigationMenu.DeleteAccount);
        });
       });
```

Файл *tests/products.spec.ts*

```
       import { test } from '@testExtenter';
       import { uiConst } from '@utils/constants/uiConst';
       test.describe(`Verify login page`, async () => {
        test.beforeEach(async ({ homePage }) => {
          await homePage.load();
        });

        test('Verify seach products by filters from home page', async ({ homePage,
caterogyProductsPage }) => {
          await homePage.selectCategory(uiConst.filters.GeneralCategories.Men,
uiConst.filters.MenCategories.Tshirts);
          await
caterogyProductsPage.verifyCategoryTitle(uiConst.filters.GeneralCategories.Men,
uiConst.filters.MenCategories.Tshirts);
          await caterogyProductsPage.verifyNumberOfProductsDisplayed(6);
          await caterogyProductsPage.verifyProductNameAndDespription('Rs. 400', 'Men
Tshirt');
        });

        test('Verify that home page displayed featured and recommended items list',
async ({ homePage }) => {
          await homePage.verifyFeaturedItemsDisplayed();
          await homePage.verifyRecommendedItemsDisplayed();
        });
```

```
        test('Add product to cart from products page', async ({ navigarationComponent,
productPage, addedToCardModal }) => {
          await navigarationComponent.clickOnMenuItem(uiConst.navigationMenu.Products);
          await productPage.verifyProductPageTitle();
          await productPage.verifySearchedProductIsDisplayed('Rs. 400', 1);
          await productPage.clickOnAddToCartButtonByIndex(1);
          await addedToCardModal.veridyModalIsVisible();
          await addedToCardModal.verifyAddedToCartMessage();
        });
        // this test has a defect - search functionality on products page is not
working correctly
        test(`Search product from products page`, async ({ navigarationComponent,
productPage }, testInfo) => {
          testInfo.annotations.push({ type: 'bug', description: 'Search functionality
on products page is not working correctly' });
          test.skip(true, 'Skipped due to bug in search functionality');
          const productName = 'Rs. 400';
          await navigarationComponent.clickOnMenuItem(uiConst.navigationMenu.Products);
          await productPage.verifyProductPageTitle();
          await productPage.searchForProduct(productName);
          await productPage.verifySearchedProductIsDisplayed(productName, 1);
        });
      });
```