Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)
Навчально-науковий інститут
комп'ютерних наук та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))
Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Магістерська кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему: «Фреймворк автоматизованого тестування вебзастосунків з використанням засобів Cypress»

Виконала: студентка 6 курсу групи КН-61м
спеціальності
122 "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)

Стецюк О.І.
(прізвище та ініціали)

Керівник Крошний І.М.
(прізвище та ініціали)

Рецензент Сторончук О.М.
(прізвище та ініціали)

Львів – 2025

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра комп'ютерних наук
Рівень вищої освіти другий (магістерський)
Спеціальність 122 "Комп'ютерні науки"
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
_____ Борецька І. Б.
"10" грудня 20 25 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Стецюк Ольги Ігорівни
(прізвище, ім'я, по батькові)

1. Тема роботи Фреймворк для автоматизованого тестування вебзастосунків з використанням засобів Cypress

керівник роботи *Крошний Ігор Миколайович, к.т.н., доцент*,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "29" квітня 2025 року № С-288
2. Термін подання студентом роботи 10 грудня 2025 року
3. Вихідні дані до роботи Розробити фреймворк для автоматизованого тестування вебзастосунків з використанням засобів Cypress. Передбачити підтримку архітектури Page Object Model, централізованих конфігурацій, фікстур тестових даних та системи звітності. Реалізувати можливість інтеграції AI-модуля для автоматичної генерації тестів за текстовим описом користувача.

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
1. Стан проблемної області
2. Інформаційне забезпечення
3. Математичне забезпечення
4. Програмне забезпечення
5. Розроблення стартап-проєкту
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Підготовка матеріалів до доповіді

6. Дата видачі завдання 1 травня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Розділ 1. Стан проблемної області	02.05.2025 – 16.05.2025	Виконано
2	Розділ 2. Інформаційне забезпечення	16.05.2025 – 16.06.2025	Виконано
3	Розділ 3. Математичне забезпечення	16.06.2025 – 04.07.2025	Виконано
4	Розділ 4. Програмне забезпечення	04.07.2025 – 18.09.2025	Виконано
5	Розділ 5. Розроблення стартап-проєкту	18.09.2025 – 15.10.2025	Виконано
6	Оформлення дипломної магістерської роботи	15.10.2025 – 27.11.2025	Виконано

Студент _____ **Стецюк О.І.**
(підпис) (прізвище та ініціали)

Керівник роботи _____ **Крошний І.М.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Магістерська робота містить 78 сторінок пояснювальної записки, 16 рисунків, 3 формули, 2 таблиці, 2 додатка, 30 джерел.

У роботі розроблено фреймворк для автоматизованого тестування вебзастосунків на основі Cypress із використанням архітектури Page Object Model, централізованих конфігурацій, тестових фікстур та модулів звітності. Додатково створено AI-модуль, що генерує тести за текстовим описом сценарію, істотно скорочуючи час автоматизації. Розроблене рішення підвищує продуктивність QA-інженерів, легко інтегрується в CI/CD процеси та може застосовуватися у промислових командах, стартапах і як навчальний інструмент у сфері тестування програмного забезпечення.

Ключові слова: Cypress, автоматизоване тестування, Page Object Model, AI-генерація тестів, тестовий фреймворк, JavaScript, end-to-end тестування**.**

ABSTRACT

The thesis contains 78 pages of explanatory notes, 16 figures, 3 formulas, 2 tables, 2 appendices, and 30 references.

This work presents the development of an automated testing framework for web applications using Cypress. The framework implements the Page Object Model architecture, centralized configurations, test fixtures, and reporting modules. Additionally, an AI module was created to automatically generate Cypress tests based on a textual scenario description, significantly reducing test development time. The proposed solution increases QA engineers' productivity, integrates seamlessly into CI/CD pipelines, and can be applied in industrial QA teams, software startups, as well as in educational programs for training software testing professionals.

Keywords: Cypress, automated testing, Page Object Model, AI-generated tests, testing framework, JavaScript, end-to-end testing.

ТЕХНІЧНЕ ЗАВДАННЯ

Необхідно розробити фреймворк для автоматизованого тестування вебзастосунків із використанням засобів Cypress, а саме:

1. Розробити архітектуру фреймворку на основі Page Object Model;
2. Створити систему централізованих конфігурацій та фікстур тестових даних;
3. Реалізувати набір кастомних команд для розширення функціональності Cypress;
4. Розробити модуль генерації звітності для відображення результатів виконання тестів;
5. Розробити та інтегрувати AI-модуль автоматичної генерації тестів за текстовим описом користувача;
6. Забезпечити можливість масштабування та інтеграції фреймворку у CI/CD процеси;
7. Провести тестування коректності роботи фреймворку та оптимізувати його структуру.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

AI – Artificial Intelligence (штучний інтелект);

API – Application Programming Interface (інтерфейс прикладного програмування) ;

CI/CD – Continuous Integration / Continuous Delivery (безперервна інтеграція та доставка) ;

CSS – Cascading Style Sheets (каскадні таблиці стилів) ;

DOM – Document Object Model (об’єктна модель документа) ;

E2E – End-to-End Testing (скрізне тестування) ;

JSON – JavaScript Object Notation (формат обміну даними) ;

HTML – Hypertext Markup Language (мова розмітки гіпертексту) ;

POM – Page Object Model (модель об’єктів сторінки) ;

QA – Quality Assurance (забезпечення якості) ;

UI – User Interface (користувацький інтерфейс) ;

URL – Uniform Resource Locator (уніфікований локатор ресурсу) ;

UX – User Experience (досвід користувача) ;

CI – Continuous Integration (безперервна інтеграція) ;

IT – інформаційні технології;

ER-модель – Entity-Relationship Model (модель «сутність–зв’язок») ;

ГБ – гігабайт;

SDK – Software Development Kit (набір інструментів розробника) ;

SaaS – Software as a Service (програмне забезпечення як сервіс).

ВСТУП

В останні роки з’явилася значна кількість наукових публікацій та практичних розробок, присвячених удосконаленню методів автоматизованого тестування вебзастосунків, підвищенню стабільності тестів та інтеграції інструментів тестування у DevOps-процеси. Особливу увагу приділено використанню сучасних фреймворків, таких як Cypress, які забезпечують високу швидкість виконання, інтерактивність, зручність налагодження та простоту інтеграції з актуальними вебтехнологіями. Паралельно активно розвиваються інструменти, що застосовують технології штучного інтелекту для автоматичного генерування тестових сценаріїв та оптимізації роботи QA-інженерів. Деякі з існуючих підходів є достатньо ефективними, але їх використання потребує комплексних рішень, здатних поєднати класичні методи автоматизації з інтелектуальними алгоритмами нового покоління.

Актуальність проблеми визначається наступними факторами:

• стрімким зростанням складності вебзастосунків та підвищенням вимог до їхньої якості, безпеки та стабільності;

• необхідністю скорочення часу тестування та підвищення продуктивності QA-фахівців в умовах швидких релізних циклів;

• активним поширенням DevOps-культури та CI/CD-підходів, що потребують високого рівня автоматизації тестування;

• популярністю сучасних фреймворків на зразок Cypress, які пропонують нові можливості для швидкого та надійного end-to-end тестування;

• зростанням інтересу до застосування штучного інтелекту в автоматизації тестування, зокрема для генерації тестів, аналізу сценаріїв та підвищення покриття;

• відсутністю універсальних рішень, що поєднують можливості Cypress із потенціалом AI-модулів та забезпечують масштабовану архітектуру для реальних командних проєктів.

Об’єктом дослідження є процес автоматизованого тестування сучасних вебзастосунків.

Предметом дослідження є методи, підходи й засоби побудови фреймворків автоматизації тестування на базі Cypress із використанням технологій штучного інтелекту для генерації тестових сценаріїв.

Метою роботи є розроблення фреймворку для автоматизованого тестування вебзастосунків із використанням засобів Cypress та інтегрованого AI-модуля для автоматичного формування тестових сценаріїв.

Для досягнення поставленої мети передбачено розв'язання таких завдань:

• проаналізувати сучасні інструменти та підходи до автоматизації тестування вебзастосунків;

• визначити вимоги до архітектури масштабованого та гнучкого фреймворку;

• розробити структуру фреймворку на основі Page Object Model із системою конфігурацій, фікстур та кастомних команд;

• реалізувати AI-модуль генерації автоматизованих тестів;

• забезпечити інтеграцію фреймворку в процеси CI/CD та організувати модуль звітності;

• провести експериментальну перевірку роботи фреймворку й оцінити ефективність запропонованого рішення.

Наукова новизна роботи полягає у створенні комплексного підходу, що поєднує можливості фреймворку Cypress із технологіями штучного інтелекту, що дозволяє автоматизувати формування тестів, зменшити обсяги ручної роботи й забезпечити динамічне розширення тестового покриття.

Практичне значення отриманих результатів полягає в тому, що розроблений фреймворк може бути використаний у командних проєктах автоматизації тестування вебзастосунків, у QA-відділах IT-компаній, у навчальному процесі для підготовки фахівців з автоматизованого тестування, а також у процесах безперервної інтеграції та розгортання.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. Аналіз сучасних підходів до автоматизації тестування вебзастосунків

Автоматизація тестування є невід'ємною складовою розроблення сучасних вебзастосунків, що дає змогу значно підвищити ефективність процесу контролю якості. Підходи до автоматизації можна класифікувати залежно від типу інструментів, архітектури та способу взаємодії з браузером.

Традиційний класичний підхід представлений фреймворками на основі Selenium WebDriver, які забезпечують взаємодію з браузером через драйвери. Він характеризується високою універсальністю, але потребує налаштування додаткових компонентів для запуску, ручної синхронізації та має нижчу швидкість виконання тестів [22].

Другий підхід – використання скриптових інструментів керування браузером через API, таких як Puppeteer та Playwright. Ці інструменти працюють швидше, забезпечують стабільніше виконання тестів та мають сучасний асинхронний механізм взаємодії з браузером. Їх використання зменшує кількість помилок синхронізації та спрощує створення тестових сценаріїв.

Окрему категорію складають інтегровані фреймворки, серед яких особливе місце посідає Cypress. Він функціонує в середовищі браузера, використовуючи внутрішній цикл виконання команд, автоматичні очікування та можливість інтерактивного перегляду тестів. Завдяки цьому Cypress забезпечує високу стабільність тестів та робить процес налагодження простішим [7, 8].

Також, новим напрямом є застосування штучного інтелекту для автоматизації тестування. Сучасні AI-інструменти здатні аналізувати поведінку користувача, опис сценарію або структуру інтерфейсу й на основі цього генерувати тестові кроки. Це підвищує покриття тестування та суттєво скорочує обсяг ручної роботи.

Проведений аналіз дає змогу зробити висновок, що сучасні підходи спрямовані на підвищення швидкості, стабільності та гнучкості тестування, а поєднання інтегрованих фреймворків із технологіями штучного інтелекту відкриває нові можливості для оптимізації цього процесу.

1.2. Огляд інструментів автоматизації тестування та їх порівняльна характеристика

Ринок інструментів автоматизації тестування представлений широким спектром рішень, що відрізняються архітектурою, швидкістю роботи та функціональними можливостями. Найпоширенішими інструментами є Selenium WebDriver, Puppeteer, Playwright, Cypress, TestCafe, WebdriverIO та Robot Framework.

Selenium WebDriver – найстаріший та найгнучкіший інструмент, що підтримує більшість мов програмування та браузерів. Основним недоліком є складність налаштування, залежність від драйверів та потреба у ручній синхронізації [22].

Puppeteer – інструмент, розроблений Google, призначений для керування браузером Chrome через DevTools Protocol. Він забезпечує високу стабільність і швидкість, проте спочатку був прив’язаний до одного браузера.

Playwright – розвиток ідей Puppeteer, який підтримує браузери Chromium, Firefox та WebKit. Має автоматичні очікування, сучасний API, зручний запуск у різних середовищах та стабільність виконання [23].

Cypress – інтегрований фреймворк, який працює всередині браузера. Він має власну архітектуру виконання команд, автоматичне очікування елементів, можливість перегляду історії виконання тестів і простий API. Основні переваги: швидкість, стабільність, зручне налагодження, детальні звіти [1 , 15].

TestCafe і WebdriverIO також використовуються в автоматизації, але мають нижчу популярність та обмеженіші можливості порівняно з Cypress і Playwright [24].

Порівняльний аналіз показує, що Cypress займає провідне місце серед інструментів для автоматизації сучасних вебзастосунків завдяки зручності, стабільності та швидкості.

1.3. Особливості та ключові переваги фреймворку Cypress

Cypress вирізняється унікальною архітектурою, яка поєднує виконання тестів у браузері з можливістю взаємодії із системою «зсередини». Це забезпечує низку переваг:

- автоматичне очікування появи елементів;
- можливість «тайм-тревелу» –перегляд кожного кроку виконання тесту;
- простий та інтуїтивний API;
- стабільність навіть у складних динамічних інтерфейсах;
- швидкий запуск великої кількості тестів;
- зручність інтеграції з CI/CD та лінійками побудови.

Фреймворк має інструменти для роботи з фікстурами даних, конфігураціями, кастомними командами та генерацією звітів, що робить його універсальною платформою для побудови складних тестових фреймворків.

1.4. Архітектурні моделі побудови фреймворків автоматизованого тестування

Одним із найпоширеніших та найефективніших архітектурних підходів до організації автоматизованих тестів є модель Page Object Model. Вона передбачає представлення кожної сторінки вебзастосунку у вигляді окремого класу, що містить локатори елементів та методи взаємодії з ними. Завдяки цьому логіка тестів відокремлюється від технічної реалізації DOM, що спрощує підтримку системи та забезпечує її масштабованість.

У межах POM легко реалізувати бізнес-логіку, повторювані операції, допоміжні модулі, а також структури конфігурацій та фікстур, які є необхідними компонентами сучасного фреймворку.

1.5. Технології штучного інтелекту в автоматизованому тестуванні

Застосування штучного інтелекту відкриває принципово нові можливості у сфері тестування. Сучасні AI-рішення можуть автоматично аналізувати інтерфейс, визначати найбільш критичні шляхи взаємодії користувача із системою та

генерувати відповідні тестові сценарії. Деякі інструменти використовують машинне навчання для відновлення тестів після оновлень, що дозволяє зменшити кількість фейкових тестів.

AI також застосовується для аналізу текстових описів користувача, перетворюючи їх на структуровані автоматизовані тести. Це істотно знижує витрати часу на ручне створення тестової документації та підвищує ефективність команд.

1.6. Аналіз аналогів AI-модулів

На ринку існують комерційні платформи, що поєднують автоматизацію тестування з технологіями штучного інтелекту. Серед них варто відзначити Testim, Functionize та Mabl. Вони можуть генерувати тести автоматично, пропонують візуальні редактори та алгоритми самовідновлення. Однак ці рішення часто мають низку обмежень, зокрема закритий вихідний код, залежність від хмарних сервісів, високу вартість та недостатню гнучкість.

У зв’язку з цим розроблення власного AI-модуля є доцільним, оскільки дозволяє повністю контролювати логіку генерації тестів, адаптувати її до потреб конкретного проєкту та забезпечити повну інтеграцію з архітектурою Cypress.

Висновок до розділу

У першому розділі було проведено всебічний аналіз проблемної області, що охоплює сучасні підходи, інструменти та технології автоматизації тестування вебзастосунків. Розглянуто основні тенденції розвитку автоматизованого тестування, а також особливості традиційних та сучасних фреймворків, які застосовуються в індустрії. Показано, що сучасні вебзастосунки стають дедалі більш складними, інтерактивними та динамічними, що зумовлює необхідність використання гнучких, стабільних і високопродуктивних рішень для тестування.

У межах аналізу було детально розглянуто інструменти автоматизації тестування, серед яких Selenium, Puppeteer, Playwright і Cypress. Проведене порівняння дозволило визначити ключові переваги та недоліки кожного із зазначених інструментів. Особлива увага була приділена фреймворку Cypress, який

завдяки своїй архітектурі та інтеграції з браузером забезпечує стабільність, високу швидкість виконання тестів, мінімальну кількість проблем синхронізації та зручність налагодження. Саме ці характеристики роблять Cypress одним із найефективніших рішень для автоматизації сучасних вебзастосунків.

Окрім цього, у розділі було досліджено можливості застосування технологій штучного інтелекту у сфері тестування, зокрема їх здатність автоматично генерувати тестові сценарії, аналізувати поведінку системи та оптимізувати тестове покриття. Аналіз наявних AI-рішень показав, що попри певні успіхи комерційних платформ, вони залишаються обмеженими у гнучкості, мають високу вартість та недостатню адаптивність під специфічні потреби проєктів. Це створює обґрунтовану необхідність у розробленні власного фреймворку, який би поєднував переваги Cypress з потенціалом штучного інтелекту.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Виділення об’єктів дослідження та їх інформаційних зв’язків

У процесі розроблення фреймворку автоматизованого тестування важливо визначити ключові інформаційні об’єкти, які беруть участь у формуванні тестових сценаріїв, їх виконанні та подальшому аналізі результатів. Фреймворк працює з декількома взаємопов’язаними сутностями, що утворюють цілісну інформаційну систему. Кожен об’єкт має свою структуру, набір атрибутів, функціональне призначення та логічні зв’язки з іншими елементами системи.

Першим основним об’єктом є сторінки вебзастосунку, які представляються у вигляді Page Object Model. Кожна сторінка розглядається як окремий логічний компонент із зафіксованою структурою, власними елементами інтерфейсу та методами взаємодії. На рівні інформаційної моделі Page Object можна трактувати як абстракцію реальної частини інтерфейсу, що містить опис елементів і набори операцій, які можна виконувати користувачем. Зокрема, об’єкт сторінки включає локатори кнопок, полів вводу, навігаційних елементів, таблиць, модальних вікон, а також методи, які описують взаємодію з цими елементами. Він виступає посередником між тестовим сценарієм і фактичним DOM, що забезпечує ізоляцію тестової логіки від структури інтерфейсу та робить систему стійкою до змін у дизайні вебзастосунку.

Наступним об’єктом є тестові сценарії. Вони описують логіку поведінки користувача, набір дій, початкові умови, очікувані результати та умови перевірки коректності виконання. Тестові сценарії є центральною одиницею автоматизованого тестування, оскільки саме вони визначають, які елементи інтерфейсу потрібно перевірити, у якій послідовності виконуються дії, які дані використовуються та за якими критеріями оцінюється успішність [29]. У фреймворку Cypress ці сценарії реалізовані у вигляді файлів коду, тоді як у контексті AI-модуля вони можуть формуватися автоматично на основі текстових описів. Кожен тестовий сценарій пов’язаний з конкретними Page Objects, оперує певними фікстурами даних,

використовує загальні конфігураційні параметри та генерує окремий набір вихідної інформації у вигляді логів та звітів [10, 16, 28].

Важливе місце займають фікстури тестових даних. Це структуровані набори інформації, найчастіше у форматі JSON, які використовуються для введення даних у вебзастосунок або для імітації реальних сценаріїв роботи користувачів. Фікстури забезпечують стабільність тестування, оскільки дозволяють відтворювати ті самі умови багаторазово, не залежачи від зовнішніх змін. У рамках інформаційної моделі фікстури пов’язані з тестовими сценаріями та можуть впливати на вибір шляхів виконання тестів. Вони містять набір атрибутів, що описують значення полів, користувацькі ролі, параметри форм, тестові набори та іншу інформацію, яка необхідна для конкретних перевірок [13,21,27,28].

Суттєвим елементом інформаційного забезпечення є конфігураційні файли, які визначають параметри роботи фреймворку. Вони містять загальні налаштування, такі як базова URL-адреса, таймаути, параметри запуску браузера, змінні середовища та інші службові значення. Ці дані забезпечують адаптивність фреймворку, оскільки дозволяють використовувати єдину структуру тестів у різних середовищах –локальному, тестовому чи продуктивному. Конфігурації також впливають на поведінку тестів, оскільки визначають способи взаємодії з інтерфейсом, час очікування подій та умови виконання перевірок [9,10].

Окрему групу формує AI-модуль генерації тестів. На відміну від статичних об’єктів, він виконує складні обчислювальні задачі, пов’язані з перетворенням текстових описів у структуровані автоматизовані сценарії. У межах інформаційної моделі AI-модуль працює з природною мовою, виконує семантичний аналіз, ідентифікує дії, елементи інтерфейсу, змінні, перевірки та пов’язує їх з відповідними Page Objects. Результатом його роботи є сформований файл тесту з чітко визначеною структурою, що відповідає архітектурі фреймворку. Таким чином, AI-модуль виступає інформаційним перетворювачем високого рівня, який створює нові об’єкти на основі вхідних даних користувача.

Завершальним об’єктом інформаційної моделі є модуль звітності та логування. Він відповідає за фіксацію інформації, яка виникає під час виконання тестів. До нього належать дані про результати проходження кожного тестового

кроку, тривалість виконання, виникнення помилок, попереджень та інші метрики. Звіти формуються у форматах HTML або JSON і можуть містити графічні знімки екрана, відеозаписи та структуровані лог-файли. Логування відіграє важливу роль у подальшому аналізі якості тестів і дозволяє відтворити причини виникнення помилок.

Сукупність зазначених об’єктів утворює складну систему інформаційних взаємозв’язків. Тестові сценарії використовують Page Objects для взаємодії з інтерфейсом, звертаються до фікстур для отримання даних, залежать від конфігураційних параметрів, можуть створюватися вручну або за допомогою AI-модуля, а результати їх виконання фіксуються у модулі звітності. Усі ці зв’язки формують цілісну інформаційну модель фреймворку, що відображає структуру, поведінку та логіку роботи системи.

2.2. Характеристика вхідних та вихідних даних фреймворку

Вхідні та вихідні дані фреймворку автоматизованого тестування визначають логіку роботи системи, особливості її функціонування та взаємодії між окремими компонентами. Коректна класифікація й опис цих даних є важливим етапом, оскільки саме вони забезпечують узгодженість процесів виконання тестів, стабільність результатів і можливість багаторазового відтворення тестових сценаріїв у різних умовах. Вхідні дані визначають, що саме система повинна протестувати, а вихідні –яким є результат цього тестування та якою мірою поведінка вебзастосунку відповідає вимогам [3].

До вхідних даних насамперед належить текстовий опис тестових сценаріїв, який може формуватися як вручну, так і за допомогою інтегрованого AI-модуля. Такий опис містить перелік дій користувача, визначення очікуваних результатів, посилання на певні елементи інтерфейсу та загальну логіку сценарію. У межах фреймворку цей текст виконує роль високорівневої моделі, яка повинна бути перетворена у структурований програмний код, що вже безпосередньо виконується Cypress. Текстовий опис часто включає інформацію про умови старту тесту,

необхідні дані користувача, можливі варіанти виконання сценарію та критерії успішності.

Важливим типом вхідних даних є фікстури тестових даних. Вони забезпечують стандартизований спосіб підготовки вхідних значень для різних тестів, що сприяє підвищенню повторюваності та надійності результатів. Фікстури зазвичай зберігаються у форматі JSON, що дозволяє зручно структурувати інформацію. У таких файлах можуть міститися значення форм, параметри запитів до API, дані користувачів різних ролей, інформація для навігації, значення, що потребують валідації, або інші залежні від контексту дані. Усі фікстури логічно пов’язані з тестовими сценаріями, які їх використовують, і часто визначають поведінку тестів у певних умовах. Використання фікстур дозволяє розділяти тестову логіку та тестові дані, що відповідає принципу модульності й робить фреймворк більш гнучким.

Ще однією важливою групою вхідних даних є конфігураційні параметри. Вони визначають середовище, у якому виконується тестування: базову URL-адресу, тип браузера, рівень логування, тривалість очікування, специфічні змінні середовища, шляхи до ресурсів та інші технічні налаштування. Конфігураційні параметри дозволяють адаптувати один і той самий набір тестів до різних середовищ, наприклад до локального запуску, тестової інфраструктури або CI/CD-конвеєра. Зміна цих параметрів не потребує модифікації тестового коду, що робить систему масштабованою та зручною у використанні.

До вхідних даних також належать структури POM, оскільки вони містять інформацію про доступні елементи вебзастосунку, можливі операції з ними та логічні залежності між компонентами інтерфейсу. Для тестового сценарію Page Object виступає джерелом інформації про те, які елементи можуть бути використані та які дії над ними доступні. Іншими словами, тест «знає» про інтерфейс саме через Page Objects.

Значну частину вхідних даних становлять також скрипти та службові модулі, які регулюють роботу фреймворку: модулі логування, хелпери для повторюваних операцій, генератори випадкових даних, утиліти для роботи з API. Хоча вони не є

даними у традиційному сенсі, вони відіграють роль інформаційних структур, що впливають на виконання сценаріїв.

Вихідні дані фреймворку представлені результатами виконання тестових сценаріїв. Найважливішим типом вихідної інформації є звіти, які створюються у форматі JSON, HTML або в інших форматах, залежно від реалізованої системи звітності. У таких звітах міститься інформація про те, які тести були виконані, які з них завершилися успіхом, які помилки виникли, яку тривалість мав кожен крок, які елементи інтерфейсу були задіяні та які саме твердження не були виконані. Звіти слугують ключем до аналізу якості тестів та їх подальшого покращення.

До вихідних даних належать також журнали виконання тестів. Логи містять детальний опис роботи фреймворку: які команди виконувалися, у якому порядку, якими були відповіді системи, які події відбувалися у браузері та на якій стадії виникли помилки. Журнали дозволяють відтворити процес проходження тесту у разі відхилення системи від очікуваної поведінки.

Відеозаписи або скриншоти, які Cypress автоматично створює при падінні тестів, також належать до вихідних даних. Візуальні матеріали спрощують аналіз проблем, оскільки дозволяють побачити фактичний стан вебзастосунку у момент виникнення помилки.

Ще одним видом вихідної інформації є сформовані тестові файли, якщо тест був створений AI-модулем. У такому випадку вихідними даними є згенерований код, що відповідає структурі фреймворку. Це дає можливість оцінити якість роботи AI-модуля та відкоригувати його алгоритми.

Таким чином, сукупність вхідних і вихідних даних визначає логіку функціонування фреймворку, забезпечує узгодженість процесів тестування та дозволяє створити гнучку й масштабовану систему, здатну працювати в різних середовищах і сценаріях використання. Вхідні дані формують основу для виконання тестів, тоді як вихідні –забезпечують аналіз якості, оцінку стабільності та можливість удосконалення фреймворку.

2.3. Інформаційна модель процесу автоматизованого тестування

Інформаційна модель процесу автоматизованого тестування відображає логічну структуру взаємодії між компонентами фреймворку та визначає порядок, у якому обробляються вхідні й вихідні дані. Вона описує систему як сукупність взаємопов'язаних елементів, що працюють у межах єдиного процесу виконання тестів. Така модель дозволяє зрозуміти, які об'єкти беруть участь у тестуванні, як вони обмінюються інформацією, які залежності між ними існують та яким чином формується кінцевий результат тестування.

Процес автоматизованого тестування можна представити як послідовність етапів, які виконуються один за одним у певній логічній структурі, де кожен етап має власні вхідні й вихідні дані [3, 6]. На початковому етапі система завантажує конфігураційні параметри, що визначають середовище виконання тестів. У межах інформаційної моделі цей етап відповідає формуванню початкового стану системи, у якому визначається базова URL-адреса, параметри браузера, таймаути, змінні середовища та інші технічні умови. Конфігурація впливає на всі подальші дії, оскільки саме вона задає контекст виконання тестів.

Наступним етапом є ініціалізація Page Object Model. На цьому рівні інформаційної моделі створюються об'єкти, які представляють логічні сторінки вебзастосунку. У кожного Page Object є власна структура даних: набір локаторів елементів, методів взаємодії та очікуваних станів сторінки. Під час виконання тесту ці об'єкти виступають джерелом інформації про інтерфейс, забезпечуючи тестовий сценарій усіма необхідними описами та операціями. Ініціалізація POM формує логічну карту інтерфейсу, яка буде використовуватися для виконання дій користувача.

Після створення Page Objects система переходить до завантаження фікстур тестових даних. На цьому етапі фреймворк обробляє JSON-файли з тестовою інформацією та формує внутрішні структури даних, які будуть використовуватися під час виконання тестових сценаріїв. У межах інформаційної моделі фікстури служать джерелом параметрів для тестів, і кожен тестовий сценарій отримує доступ до тих даних, які необхідні для його коректного виконання. Якщо тест описує

сценарій авторизації, він отримує дані користувача; якщо передбачено роботу з формами, тест одержує значення полів; якщо потрібно виконувати API-запити, надаються відповідні параметри.

Сформувавши початкові параметри середовища, структури сторінок та тестові дані, система переходить до виконання тестових сценаріїв. У межах інформаційної моделі тестовий сценарій представляється як послідовність команд, які використовують Page Objects, фікстури та конфігурації для взаємодії з вебзастосунком. Кожна команда тесту сприймається як інформаційний запит, який надсилається до відповідного Page Object. Той, у свою чергу, визначає спосіб взаємодії з елементом DOM та виконує необхідну операцію, наприклад натискання кнопки або введення тексту.

Під час виконання тесту постійно відбувається обмін інформацією між фреймворком, браузером та вебзастосунком. Cypress автоматично відслідковує зміну станів DOM, виконує очікування появи елементів, перехоплює відповіді серверів та аналізує результати взаємодії. У рамках інформаційної моделі це представляє собою механізм зворотного зв’язку, в якому кожна команда супроводжується отриманням певної інформації про фактичний стан системи.

Після завершення виконання кожного кроку тесту система фіксує його результат у внутрішній структурі логування. Логи містять інформацію про успішність виконання, тривалість кроку, фактичні значення елементів, відповіді серверів та можливі помилки. На рівні інформаційної моделі ці дані становлять окремий інформаційний потік, який надалі буде використаний для формування звітів.

Після завершення виконання всіх тестових сценаріїв фреймворк переходить до генерації звітів. На цьому етапі система обробляє результати логування, узагальнює їх та формує вихідні структуровані документи. HTML-звіти дозволяють переглядати історію виконання тестів у графічному вигляді, тоді як JSON-звіти забезпечують можливість автоматизованої обробки даних іншими інструментами. У рамках інформаційної моделі це означає завершення життєвого циклу тестів та формування вихідного набору даних, який відображає якість виконання та відповідність роботи вебзастосунку заданим вимогам [5 ,11].

Узагальнюючи, інформаційна модель процесу автоматизованого тестування описує систему як послідовність взаємодій між конфігураціями, Page Objects, фікстурами, тестовими сценаріями та модулем звітності. Вона демонструє, як дані переходять від однієї структури до іншої, як побудовані логічні зв'язки між елементами фреймворку та яку роль відіграє кожен компонент у забезпеченні коректності, стабільності та повторюваності автоматизованого тестування.

2.4. Концептуальна модель фреймворку

Концептуальна модель фреймворку автоматизованого тестування визначає його загальну структуру, внутрішню організацію та взаємозв'язки між основними компонентами системи. Вона формує абстрактне уявлення про архітектуру фреймворку, описує його логічні елементи, функціональні зони та ті інформаційні потоки, які виникають у процесі виконання тестів. Концептуальна модель є незалежною від конкретної реалізації, формату зберігання даних або мови програмування. Її завдання полягає в тому, щоб показати, як працює система в цілому, які функції виконує кожен компонент та яким чином забезпечується узгодженість усіх процесів [9, 11, 12, 14].

У центрі концептуальної моделі знаходиться тестовий рушій, який відповідає за планування, виконання та координацію тестових сценаріїв. Він є основним керівним компонентом, який ініціює всі дії, взаємодіє з іншими частинами фреймворку та формує основний інформаційний потік. Тестовий рушій отримує сценарії, інтерпретує їх, готує середовище, ініціалізує Page Objects, обробляє фікстури, звертається до функцій AI-модуля, а після завершення –передає результати модулю звітності. У межах концептуальної моделі він виступає центральним вузлом, до якого під'єднані всі інші компоненти.

Надзвичайно важливим елементом моделі є Page Object Model –компонент, який представляє структуру вебзастосунку у вигляді набору об'єктів сторінок. Кожен Page Object є окремою сутністю, що містить описи елементів інтерфейсу, методи взаємодії з ними та очікувані стани сторінки. Page Objects використовуються тестовим рушієм у процесі виконання сценаріїв, забезпечуючи логічний і

структурований доступ до інтерфейсу. У концептуальній моделі ці об’єкти займають центральне місце у взаємодії з тестами, оскільки визначають спосіб виконання дій на сторінках і значною мірою впливають на стабільність і повторюваність результатів[4, 15, 16, 28].

Ще однією ключовою складовою концептуальної моделі є модуль тестових сценаріїв. На концептуальному рівні він містить набір інструкцій, які описують поведінку тестованого вебзастосунку. На відміну від конкретної реалізації у вигляді файлів, у концептуальній моделі тестовий сценарій розглядається як логічна структура, що складається з послідовності дій, перевірок і переходів між різними станами системи. Цей модуль тісно пов’язаний із Page Objects, оскільки кожна дія та кожна перевірка реалізуються через методи відповідних сторінок.

Окреме місце у концептуальній моделі займає AI-модуль генерації тестів. Він виконує роль інтелектуального перетворювача, здатного приймати на вхід природномовні описи сценаріїв і перетворювати їх на формалізований набір інструкцій. У концептуальній моделі цей модуль взаємодіє з текстовими описами, аналізує зміст фраз, визначає ключові дії, співвідносить їх з Page Object Model та формує новий тестовий сценарій у структурованому вигляді. Таким чином, AI-модуль забезпечує можливість автоматичного створення тестів, що суттєво розширює функціональність фреймворку.

Ще одним компонентом концептуальної моделі є система обробки тестових даних. Вона відповідає за підготовку, зберігання та передачу фікстур, які використовуються під час виконання тестів. Функції цього модуля включають роботу з форматами даних, забезпечення доступу тестів до потрібної інформації, керування залежностями між тестами та дотримання узгодженості даних у межах одного сценарію. У концептуальній моделі цей компонент пов’язаний із тестовими сценаріями, оскільки кожен тест може потребувати власного набору вхідних значень.

Завершальним елементом концептуальної моделі є модуль звітності та логування, який фіксує результати виконання тестів. У концептуальному вигляді цей модуль описується як структура, що отримує інформацію від тестового рушія та формує підсумкові дані про роботу фреймворку. Сюди входить збір інформації про

виконані кроки, часові характеристики, помилки, стан інтерфейсу в момент збою, а також загальна статистика по всій тестовій сесії. Модуль звітності відповідає за створення структурованих документів, зазвичай у вигляді HTML- або JSON-звітів, які надалі використовуються для аналізу.

Усі зазначені компоненти взаємодіють між собою через інформаційні зв'язки. Тестовий рушій об'єднує роботу Page Objects, тестових сценаріїв, фікстур, AI-модуля та модуля звітності в єдиний процес. Page Objects забезпечують доступ до інтерфейсу, сценарії визначають логіку роботи, фікстури забезпечують дані, AI-модуль створює нові сценарії, а модуль звітності завершальний етап –фіксацію результатів. Ці взаємозв'язки дозволяють представити фреймворк як комплексну систему, у якій кожен компонент виконує власні функції, але водночас залежить від інших елементів.

Таким чином, концептуальна модель фреймворку демонструє цілісну структурну картину системи. Вона описує ключові компоненти, визначає логіку їх взаємодії, окреслює інформаційні потоки та формує абстрактне уявлення про архітектуру рішення. Така модель є фундаментом для подальшого розроблення логічної моделі, вибору архітектурних рішень, визначення програмних інтерфейсів і створення детального проєкту фреймворку.

2.5. ER-модель структури тестових даних та конфігурацій

ER-модель (Entity–Relationship model) є важливим засобом концептуального аналізу даних у системах, де структура інформації має ключове значення для функціонування програмного забезпечення. У контексті розроблення фреймворку автоматизованого тестування ER-модель дозволяє описати логічні зв'язки між тестовими даними, конфігураційними параметрами та іншими інформаційними сутностями, які беруть участь у виконанні тестових сценаріїв. Оскільки тестовий фреймворк працює з великою кількістю різнорідних даних –від локаторів елементів до параметрів середовища –побудова чіткої моделі таких даних є важливою умовою його стабільності, масштабованості та підтримуваності.

У системі автоматизованого тестування тестові дані виступають базовою сутністю, яка визначає поведінку сценаріїв та впливає на результати виконання тестів. Кожен тест використовує певні набори інформації, які можуть бути у вигляді JSON-фікстур, параметрів запитів, облікових записів користувачів, значень форм або системних атрибутів. У межах ER-моделі тестові дані розглядаються як окрема логічна сутність, яка має власні атрибути, структуру та контекст використання. Наприклад, дані користувача можуть містити атрибути «ім’я», «роль», «електронна адреса», «пароль», тоді як дані для API-запитів можуть включати «ідентифікатор ресурсу», «значення параметра», «очікуваний статус відповіді».

Ці дані мають важливу властивість –вони є незалежними від конкретних тестових сценаріїв, але використовуються багатьма з них. Це означає, що в ER-моделі «тестовий сценарій» і «фікстура» знаходяться у відношенні «багато-до-багатьох»: один сценарій може використовувати кілька фікстур, а одна фікстура може обслужити десятки сценаріїв. Такий зв’язок свідчить про необхідність централізованого зберігання та доступу до тестових даних, що значно полегшує підтримку фреймворку.

Іншою сутністю ER-моделі є «конфігурація». Конфігураційні дані визначають параметри, за якими виконується тестування: URL-адресу середовища, налаштування браузера, рівень логування, параметри часу очікування, значення змінних середовища та інші службові характеристики. Це означає, що конфігурація виступає незалежною сутністю, яка впливає на всі інші об'єкти моделі. У структурі ER-моделі вона має зв’язки з тестовими сценаріями, оскільки сценарій виконується з використанням певного набору конфігураційних параметрів; з Page Object Model, оскільки може визначати специфічні змінні, що впливають на логіку сторінки; та з модулем запуску тестів, який інтерпретує параметри конфігурації перед виконанням сценарію.

Тестовий сценарій, у свою чергу, виступає сутністю, що поєднує багато елементів ER-моделі. Він взаємодіє з тестовими даними, конфігураціями, Page Objects, AI-модулем та модулем звітності. У контексті ER-моделі тестовий сценарій можна представити як складну сутність, що містить «кроки тесту», «перевірки», «послідовність взаємодій» та «очікувані результати». Кожен крок тесту має власні

атрибути: дію, елемент взаємодії, очікувані значення та умови завершення. Це дозволяє розглядати тест як набір взаємопов’язаних внутрішніх сутностей, кожна з яких має відношення до Page Objects.

Page Object, своєю чергою, є сутністю, що описує структуру сторінки. Він містить локатори, методи взаємодії та інші характеристики. У межах ER-моделі Page Object має зв’язок з локаторами, які можна розглядати як окремі сутності, що містять атрибути: тип елемента, унікальний шлях, додаткові критерії пошуку, очікувані стани. Такі зв’язки дозволяють описати POM як складну структуру, де кожна сторінка складається з певного набору елементів, які у свою чергу можуть бути використані багатьма тестовими сценаріями.

Конфігурації також входять у зв’язок із модулем AI-генерації тестів. Це пов’язано з тим, що AI-модуль повинен знати контекст середовища, для якого генерується тест. Якщо, наприклад, у конфігурації визначено інший базовий URL, AI-модуль має врахувати це під час побудови структури тесту. Таким чином, навіть інтелектуальне перетворення даних відбувається у межах ER-моделі, яка впорядковує потоки інформації.

Ще одним важливим компонентом ER-моделі є модуль звітності. Хоча звіти генеруються вже після завершення тестування, вони базуються на сутностях, що з’являються у процесі роботи. Звіт містить інформацію про сценарії, їх кроки, дані, конфігурації, Page Objects, відповіді системи та фактичні результати. Тому у межах ER-моделі звіт є похідною сутністю, яка має зв’язки з усіма ключовими об’єктами: тестами, даними, конфігураціями, локаторами та станами сторінок.

Уся ER-модель утворює складну мережу зв’язків, де кожен компонент фреймворку впливає на інші. Тестові сценарії залежать від фікстур та конфігурацій; фікстури залежать від структури даних; Page Objects залежать від інтерфейсу вебзастосунку; AI-модуль залежить від змісту тестового опису; звітність залежить від тестів. Така модель забезпечує глибоке розуміння того, як саме обробляються дані, як виникають залежності та в який спосіб досягається коректність і узгодженість усього фреймворку.

2.6. Обмеження та припущення щодо організації вхідних даних та виконання тестів

Будь-яка програмна система, зокрема фреймворк автоматизованого тестування, функціонує в межах певних технічних, логічних, архітектурних та експлуатаційних обмежень. Їхнє чітке визначення є необхідною умовою для побудови правильної інформаційної моделі, оскільки саме обмеження визначають можливості системи, її поведінку, потенційні помилки та межі масштабованості. Крім того, припущення щодо організації вхідних даних і виконання тестів відіграють ключову роль у забезпеченні стабільності тестового середовища, відтворюваності результатів та узгодженості всіх процесів.

Одним із основних обмежень є залежність роботи фреймворку від коректності та повноти вхідних даних. Усі тестові сценарії, які генеруються або виконуються системою, ґрунтуються на текстових описах, Page Objects, фікстурах і конфігураціях. Якщо хоча б один із цих об’єктів містить помилку або описаний неповністю, це впливає на можливість виконання тесту. Наприклад, некоректний текстовий опис може призвести до того, що AI-модуль неправильно інтерпретує дії користувача або сформує неповний набір тестових кроків. Аналогічно, фікстури, які містять неповні або недійсні дані, призводять до неможливості виконання певних операцій у вебзастосунку. Це означає, що стабільність фреймворку залежить від того, чи є його вхідні дані структурованими, несуперечливими та логічно узгодженими.

Другим важливим обмеженням є специфіка роботи Cypress у підтримуваних браузерах. Cypress виконує тести всередині браузера, маючи повний контроль над подіями DOM, проте працює лише з певним набором браузерів –Chromium, Chrome, Edge та частково Firefox. Це означає, що тестовий фреймворк може бути обмежений у сценаріях, які потребують специфічної поведінки інших браузерів, наприклад Safari або Internet Explorer. Таке обмеження впливає на можливість тестування міжбраузерної сумісності та може вимагати додаткових засобів у тому випадку, якщо вебзастосунок має критичні функції, прив’язані до конкретних браузерів [7, 27, 30].

Обмеження також накладаються з боку самого вебзастосунку. Якщо інтерфейс часто змінюється, Page Objects можуть втрачати актуальність, а тести –падати навіть у разі відсутності фактичних дефектів. Саме тому важливо припускати стабільність DOM або, принаймні, передбачати регулярне оновлення Page Objects. Частий рефакторинг інтерфейсу може створювати значне навантаження на підтримку фреймворку, якщо система не має достатнього рівня гнучкості.

Особливу увагу слід приділити обмеженням, які стосуються роботи AI-модуля. Його ефективність залежить від якості вхідних текстових описів, а також від повноти наявної інформації щодо структури вебзастосунку. Якщо тестовий сценарій описано нечітко, а формулювання є неоднозначним, штучний інтелект може неправильно інтерпретувати необхідні дії. Крім того, AI-модуль може обмежуватися певними алгоритмічними моделями, які не враховують нестандартні або складні сценарії, для яких потрібна ручна розробка тестів. Це означає, що фреймворк має працювати з припущенням, що AI-модуль не замінює повністю людського аналітика, а лише допомагає прискорити створення типових тестів.

Суттєвим обмеженням є залежність тестового середовища від мережевої інфраструктури або від стану серверів, до яких звертається вебзастосунок. Нестабільність мережі, повільні відповіді або тимчасові збої в API можуть спричиняти помилкові падіння тестів. У межах інформаційної моделі ці явища вважаються зовнішніми факторами, але вони впливають на весь життєвий цикл тестування. Саме тому під час планування тестів важливо передбачити припущення про доступність серверів, стабільність API та консистентність зовнішніх даних.

У системі також існують логічні обмеження, які стосуються структури Page Objects. Якщо структура сторінок побудована неправильно, із надмірною кількістю залежностей або дублюванням елементів, фреймворк може ставати нечітким або важким для підтримки. Це формує припущення про те, що Page Object Model повинна бути побудована відповідно до архітектурних принципів, із чітким поділом відповідальностей, ієрархією сторінок та логічним розподілом методів.

Крім того, до обмежень належить потреба в однакових умовах виконання тестів. Результати автоматизованих сценаріїв повинні бути відтворюваними. Проте деякі фактори (динамічні елементи інтерфейсу, час завантаження сторінок, змінні

API) можуть впливати на стабільність виконання. Це означає, що тестове середовище повинно бути максимально стандартизованим, а тестові дані – незмінними у межах одного циклу тестування.

Усі зазначені обмеження та припущення формують важливий контекст для побудови інформаційного забезпечення фреймворку. Вони визначають, у яких межах система здатна забезпечувати коректну роботу, як обробляються дані, які об’єкти можуть змінюватися та в який спосіб слід організовувати тестову архітектуру, щоб забезпечити її стабільність та ефективність. Розуміння цих обмежень дозволяє не лише передбачити потенційні ризики, але й сформувати оптимальну концепцію побудови фреймворку, який здатний працювати у реальних умовах розроблення та експлуатації вебзастосунків.

Висновок до розділу

У другому розділі було проведено комплексне дослідження інформаційного забезпечення фреймворку автоматизованого тестування вебзастосунків із використанням Cypress та інтегрованого модуля штучного інтелекту. Розглянуто основні інформаційні об’єкти, що формують основу системи: тестові сценарії, структури Page Object Model, фікстури тестових даних, конфігураційні параметри, AI-модуль та модуль звітності. Кожен із цих компонентів має власний набір атрибутів, функцій і зв’язків, які у сукупності створюють єдину інформаційну модель фреймворку.

Побудована інформаційна модель процесу автоматизованого тестування демонструє, що функціонування фреймворку є багаторівневим та послідовним, а всі його частини перебувають у постійній взаємодії. Було показано, що тестування починається з формування середовища, завантаження конфігурацій, підготовки Page Objects та фікстур, після чого система переходить до виконання тестових сценаріїв. Кожен етап супроводжується обробкою даних, зворотними зв’язками та фіксацією результатів, що забезпечує можливість формування детального звіту після завершення тестування.

У рамках розділу також було проаналізовано концептуальну модель фреймворку, яка описує його архітектурну структуру. Визначено ролі ключових компонентів та характер їх взаємодії, що дає змогу розглядати систему як комплексне рішення, здатне забезпечувати стабільність, масштабованість і ефективність автоматизованого тестування. Особливу увагу приділено ER-моделі, яка показала логічні зв’язки між тестовими даними, конфігураціями, тестовими сценаріями та структурою інтерфейсу вебзастосунку.

Окремо були визначені обмеження та припущення, які впливають на якість виконання тестів і формування тестових даних. Розуміння цих обмежень дає змогу враховувати особливості роботи фреймворку, запобігати можливим помилкам і підвищувати надійність автоматизованих перевірок.

Таким чином, проведений інформаційний аналіз підтверджує, що побудова фреймворку автоматизованого тестування потребує чіткої структури даних, продуманої архітектури взаємодії компонентів та врахування специфічних обмежень середовища. Сформована інформаційна база є фундаментом для подальшого математичного моделювання, розроблення алгоритмів і реалізації програмної частини фреймворку, що буде розглянуто у наступному розділі.

РОЗДІЛ 3. МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Математична модель тестування

Автоматизоване тестування у середовищі Cypress можна розглядати як процес керованої взаємодії з вебзастосунком, де кожна дія тестового сценарію зумовлює зміну стану системи. У цьому сенсі тестування має знакову математичну особливість: воно не є випадковим набором кроків, а являє собою ланцюг перетворень на множині станів. Це дозволяє будувати формальні моделі, які описують поведінку інтерфейсу та логіку зміни DOM після кожної взаємодії.

Вебзастосунок SwagLabs, що використовується у дипломній роботі, має чітку структуру сторінок та переходів. Кожна сторінка містить власні елементи й набір можливих дій, які змінюють систему (рис. 3.1). Стан системи також включає такі параметри, як авторизація користувача, вміст корзини, обрана сортування товарів, введені користувачем дані та контекст взаємодії [18]. Складність полягає в тому, що ці стани можуть змінюватися не тільки під дією тесту, а й через внутрішні механізми браузера: асинхронні запити, рендеринг, анімації.

Cypress перетворює хаотичний браузерний світ на детерміновану модель. Він очікує доступність елементів, повторює дії при необхідності, синхронізує всі команди з DOM. Це створює основу, завдяки якій тест може бути інтерпретований як математична композиція функцій. Формально процес тестування можна описати як дискретну модель переходів станів:

$$S_{t+1} = \delta(S_t, a_t), \tag{3.1}$$

де S_t — поточний стан системи, a_t — дія Cypress, δ — функція переходу, S_{t+1} — новий стан після виконання дії.

Подібна модель дозволяє зрозуміти, що будь-який тест не просто виконує інструкції, а керує процесом перетворення системи у відповідності до заданої цільової конфігурації –наприклад, користувач успішно оформив замовлення.

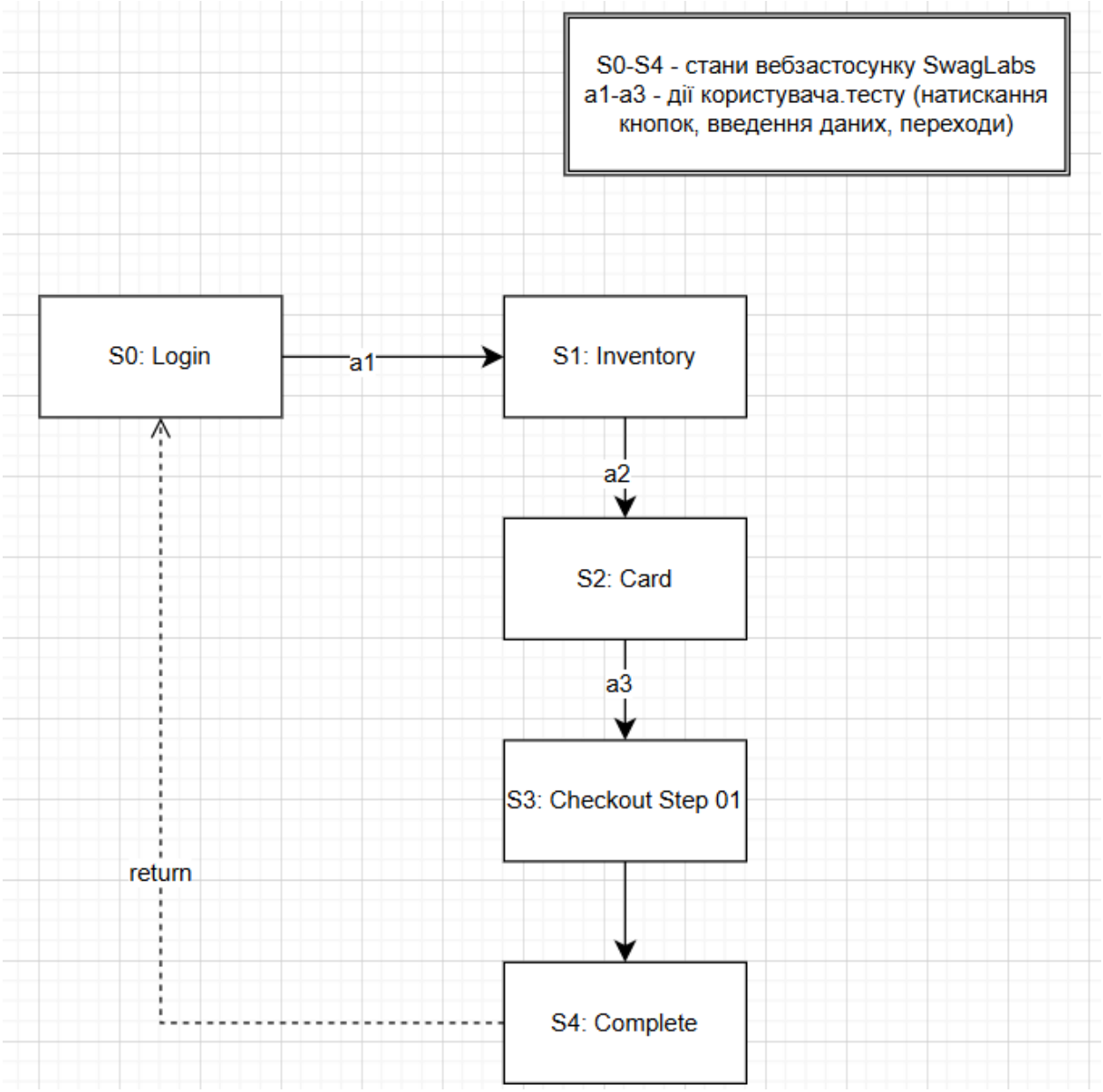

Рисунок 3.1 – Математична модель тестування

Окремої уваги заслуговує принцип причинно-наслідкової визначеності. Якщо Cypress не може знайти елемент, не може на нього натиснути, або елемент тимчасово недоступний, дія не буде виконана, і тест не зможе перейти у наступний стан. У класичних алгоритмах це відповідало би порушенню умови переходу. Тому можна сказати, що Cypress “захищає модель” від переходів у некоректні або невалідні стани. Саме ця властивість дозволяє формалізувати тестування як послідовність строго визначених, перевірених та узгоджених змін.

3.2. Математична модель Page Object Model

Page Object Model у межах даної роботи розглядається не лише як шаблон організації коду, але і як абстрактна модель сторінки, що дозволяє розділити бізнес-логіку тесту та технічну логіку взаємодії з DOM. Кожен Page Object описує сторінку як математичний об’єкт із власними властивостями (елементами), поведінкою (методами) та обмеженнями (очікуванням певного стану).

У фреймворку такі об’єкти представлені у вигляді класів LoginPage, InventoryPage, CartPage, CheckoutPage. Кожен з них містить селектори та методи, які не лише взаємодіють з DOM, але й відповідають за логічні переходи між станами вебзастосунку. Наприклад, метод loginAs у LoginPage є композицією низки атомарних дій: відкриття сторінки, введення логіна, введення паролю та натискання кнопки. Таким чином, Page Object можна формально подати як множину елементів сторінки:

$$P = \{e_1, e_2, \dots, e_n\}, \tag{3.2}$$

де P — сторінка (Page Object), $e_1 \dots e_2$ — елементи сторінки.

Кожна з цих дій сама по собі є функцією, що змінює стан системи. Таким чином, Page Object виступає механізмом створення складних функцій із простих – повна відповідність концепції композиції у математиці (рис. 3.2).

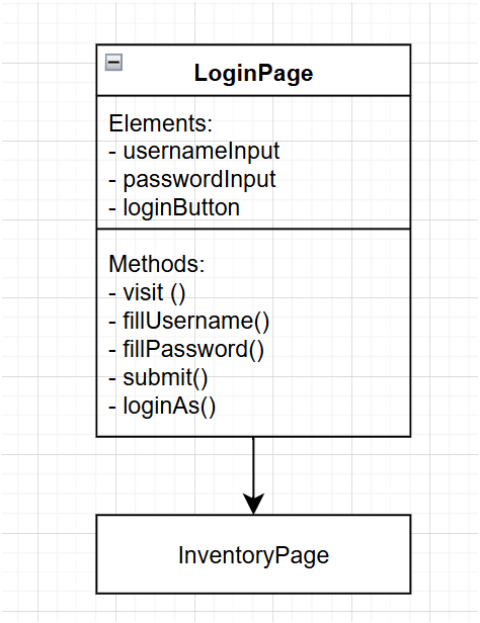

Рисунок 3.2 – Приклад Page Object Model сторінки авторизації

POM також забезпечує формальну інваріантність: якщо визначено, що метод addToCart у InventoryPage додає товар у корзину, то це твердження залишається істинним незалежно від сценарію, який його викликає. Це робить Page Object стабільною математичною сутністю з передбачуваною поведінкою. У тестуванні це критично, адже фреймворк повинен забезпечувати можливість розширення і змін без порушення логічних взаємозв'язків.

З точки зору теорії алгоритмів Page Object дає змогу будувати складні маршрути тестування як комбінацію алгоритмів окремих сторінок. У випадку SwagLabs логін, додавання товарів у корзину, оформлення замовлення та валідація результатів формуються через низку викликів методів Page Object. Тому POM у цій роботі виступає не просто патерном, а елементом математичного моделювання поведінки вебзастосунку.

3.3. Алгоритм роботи Cypress

Робота Cypress з точки зору алгоритмів є прикладом побудови інтерпретованої системи управління браузером. Тестовий файл не виконується одразу, а спершу аналізується, і всі команди додаються у внутрішню чергу. Це дозволяє моделювати тест як формальну програму, що складається з послідовності команд, де кожна команда має власний життєвий цикл: пошук елемента, перевірка стану, виконання дії, контроль результату [4, 7, 20].

Автоматичне очікування (auto-wait) є ключовим елементом алгоритму Cypress. Замість того, щоб одразу виконувати команду, Cypress повторює спробу доти, доки елемент не стане доступним або не мине максимальний час очікування. Це робить поведінку тесту гнучкою та стійкою до тимчасових затримок у рендерингу сторінки. У разі відсутності елемента Cypress не переходить до наступної команди, що забезпечує строгість виконання алгоритму та унеможливлює появу помилок, пов'язаних із надто швидким виконанням тестів (рис. 3.3).

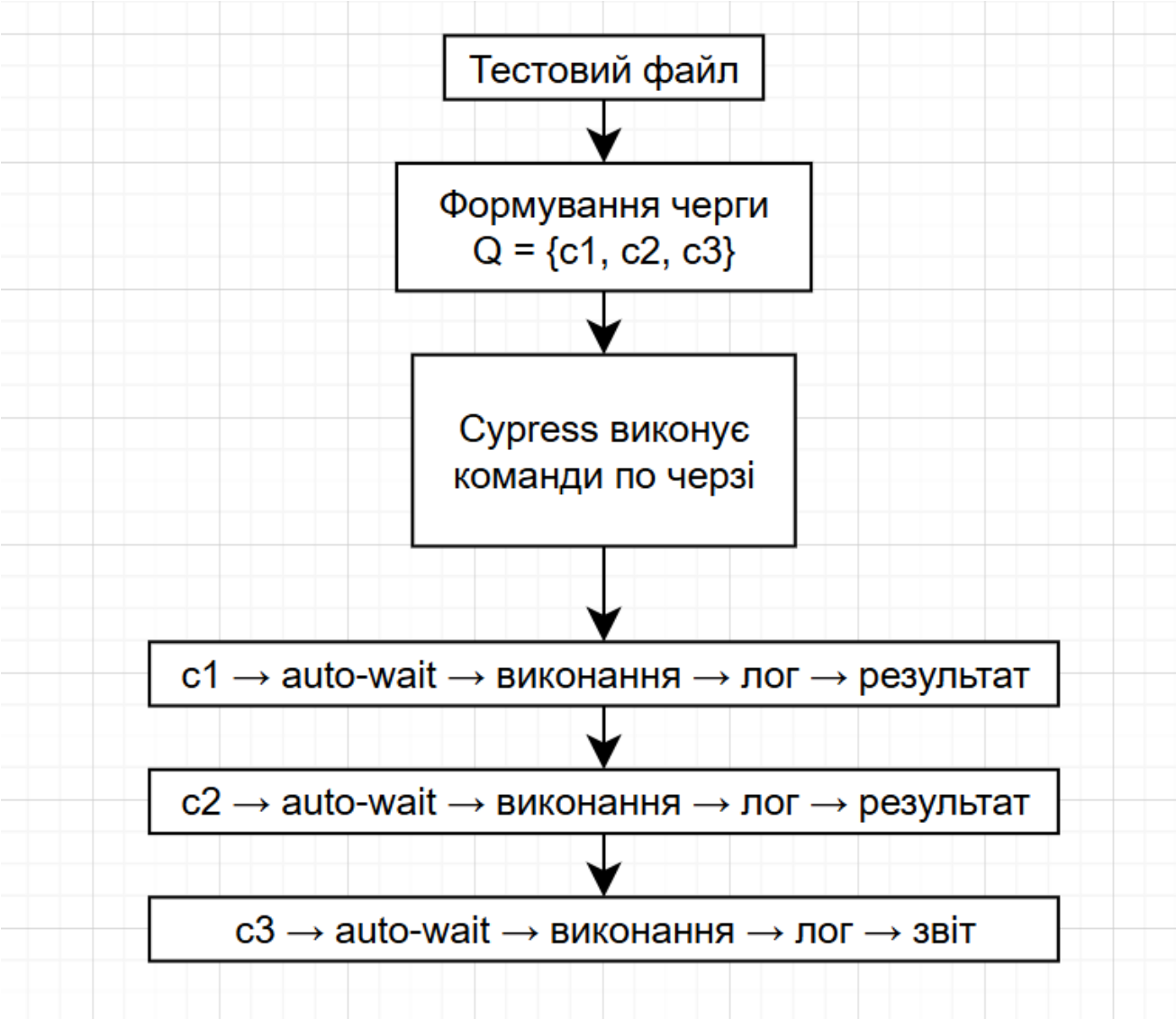

Рисунок 3.3 – Алгоритм роботи Cypress (черга команд)

У випадку отримання помилки Cypress негайно фіксує контекст: створює скріншот, зберігає HTML-структуру, записує послідовність виконаних команд. Цей процес можна інтерпретувати як алгоритм аварійного переривання, який дозволяє коректно завершити тест навіть у разі некоректної поведінки браузера. Після завершення всіх тестів Cypress формує детальний звіт через Mochawesome, який включає інформацію про час виконання кожної команди, їхній статус, повідомлення про помилки та допоміжні артефакти [19].

Таким чином Cypress є прикладом складної інтерпретованої системи, що працює за принципами синхронного послідовного виконання, автоматичного відновлення після помилок та строгого контролю черги команд.

3.4. Алгоритм роботи AI-модуля

AI-модуль, реалізований у дипломній роботі, працює як інтелектуальний інтерфейс між природною мовою та формальним тестовим кодом. Він аналізує текстовий опис користувацького сценарію і перетворює його на Cypress-тест, що відповідає структурі проєкту. Це дозволяє розглядати алгоритм роботи модуля як трансформацію даних із семантичного простору у простір формальних команд.

У роботі модуль складається з Node.js сервера, який отримує запит від користувача та створює prompt для моделі штучного інтелекту. Prompt включає інструкції, приклад тесту та конкретні вимоги щодо структури і синтаксису. Таким чином система визначає цільовий формат, і AI генерує код, який вже відповідає правилам фреймворку (рис. 3.4).

Рисунок 3.4 – Алгоритм AI-модуля

Отриманий тестовий сценарій містить реальні Cypress-команди, які взаємодіють з DOM, використовують Page Object Model або прямі селектори. Після генерації коду сервер створює файл у папці cypress/e2e/ai. Далі цей тест може бути виконаний так само, як і всі інші.

Алгоритм AI-модуля розширює можливості фреймворку, оскільки автоматизує початковий етап написання тесту та знижує імовірність людських помилок. Крім того, він демонструє, що сучасні мовні моделі можна інтегрувати у класичні інструменти тестування, створюючи гібридні рішення.

3.5. Формалізація тестових даних

Тестові дані відіграють ключову роль у забезпеченні повторюваності та стабільності тестів. Використання фікстур дозволяє зберігати дані у незалежних JSON-файлах і робить тести універсальними: тестовий код не залежить від конкретних значень. Це дає змогу виконувати однакові сценарії для різних наборів даних або швидко оновлювати вміст тестів без зміни їхньої логіки.

Під час тестування SwagLabs у фікстурах зберігалися дані користувачів, товари для корзини та дані покупця. Cypress завантажує фікстури на початку кожного тесту, перетворюючи JSON у об’єкти JavaScript, що дозволяє легко використовувати їх у Page Object методах. Тестові дані можна розглядати як множину параметрів, що визначають початковий стан системи та її цільові очікування.

Фікстури також допомагають ізолювати тести один від одного. Оскільки кожен тест отримує власні дані, його поведінка не залежить від попередніх сценаріїв або сторонніх змін. Цей підхід сприяє підвищенню стабільності та прогнозованості.

3.6. Граф переходів станів

Під час тестування користувач проходить послідовність кроків: логін, перегляд товарів, додавання їх у корзину, оформлення замовлення та підтвердження покупки. Цю поведінку можна подати як граф переходів (рис. 3.5), де кожна вершина є окремою сторінкою, а перехід –дією користувача або тесту.

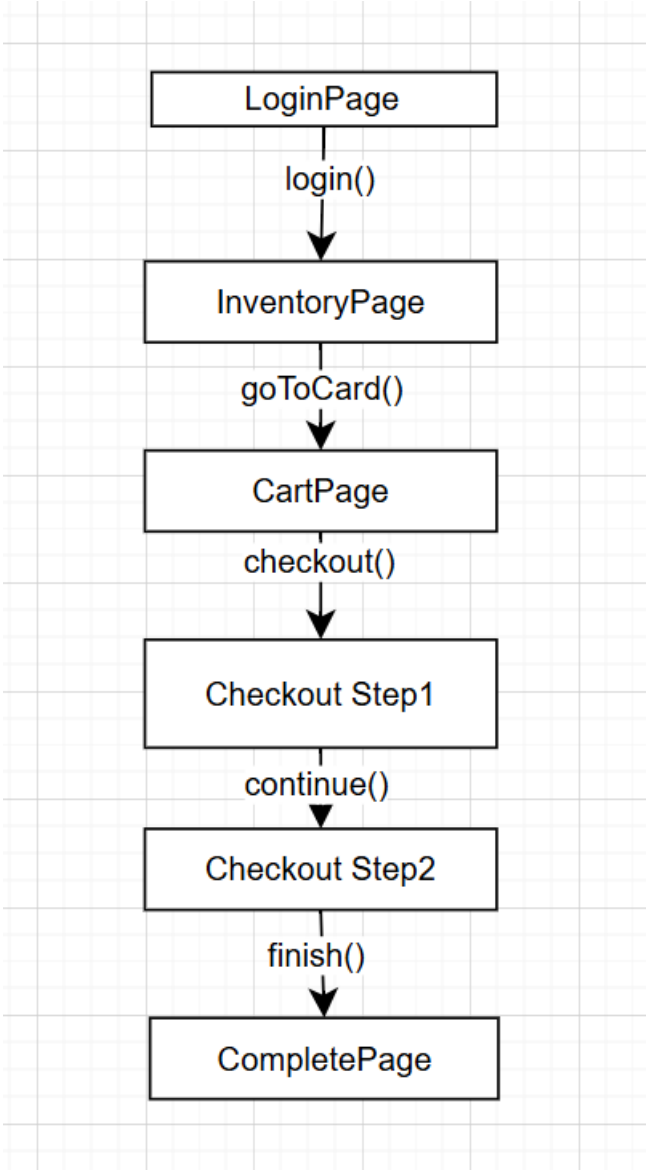

Рисунок 3.5 – Схема переходів між сторінками SwagLabs

Такий граф допомагає побачити логіку системи як набір станів, що послідовно змінюються. Він також дозволяє визначити всі можливі сценарії тестування, включно з альтернативними маршрутами: неправильний пароль, порожня корзина, неповне заповнення даних тощо. У розробленому фреймворку використання цього графа дає змогу структурувати Page Object та визначити чітку логіку переходів, яку AI-модуль може використовувати для генерації тестів.

3.7. Теоретичні засади стабільності тестів

Стабільність тестів залежить від того, наскільки детермінованою є поведінка інтерфейсу та наскільки правильно тестовий фреймворк синхронізує свої дії з браузером. Cypress забезпечує цю стабільність завдяки вбудованим механізмам автоматичного очікування, повторів та контролю асинхронності [25. 26].

Однак не менш важливою є стабільність, що створюється на рівні тестового фреймворку. У розробленому рішенні POM стандартизує взаємодію з елементами DOM, забезпечуючи можливість повторного використання методів. Це мінімізує ризик помилок, пов’язаних зі зміною селекторів або бізнес-логіки.

AI-модуль також впливає на стабільність, оскільки генерує тести у єдиному стилі. Це усуває різницю між тестами, написаними вручну різними людьми, та сприяє уніфікованості фреймворку. Таким чином, стабільність тестів у межах цієї роботи можна виразити простою моделлю:

$$S = C + D + U, \tag{3.3}$$

де S – стабільність, C – коректність роботи Cypress, D – стабільність та ізольованість тестових даних, U – уніфікованість структури тестів (AI + POM).Тому стабільність тестів у цій роботі визначається як поєднання коректної реалізації Cypress-алгоритмів, ізольованості тестових даних та уніфікованих сценаріїв взаємодії з інтерфейсом (рис. 3.6).

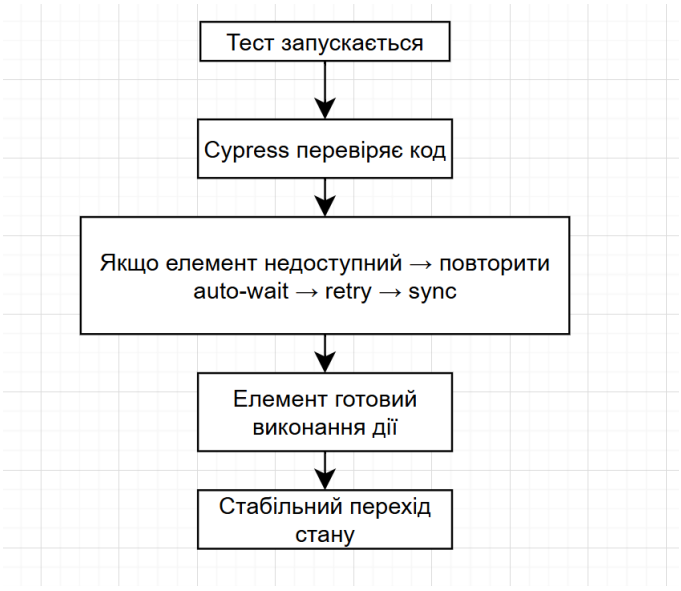

Рисунок 3.6 – Модель стабільності тестів

Висновок до розділу

У третьому розділі було здійснено комплексний математичний та алгоритмічний аналіз фреймворку автоматизованого тестування, розробленого в межах даної дипломної роботи. Були побудовані формальні моделі процесу тестування як послідовності переходів між станами вебзастосунку SwagLabs, що дозволило представити тест не як набір випадкових команд, а як детермінований механізм зміни системи. Описано Page Object Model як абстракцію сторінки, що об’єднує елементи, логіку поведінки та алгоритми переходів між станами. Показано, що POM не лише структурує код, але й створює математичну модель поведінки користувача.

Алгоритмічні принципи Cypress, включно з механізмами формування черги команд, автоматичного очікування, повторних спроб та логування, були розглянуті як складна система синхронізації дій із DOM, що забезпечує стабільність та відтворюваність тестів. Окремо було досліджено роботу AI-модуля, який виконує перетворення природної мови в формальний тестовий код, та проаналізовано його роль як елемента інтелектуальної автоматизації. Було розглянуто концепцію тестових даних як структурованої множини параметрів, що впливають на поведінку тестів, та побудовано граф переходів між сторінками вебзастосунку.

У результаті математичне забезпечення дозволило описати фреймворк як цілісну систему, де кожен компонент –Cypress, POM, фікстури, AI-модуль – взаємодіє у межах чітко визначених алгоритмів і залежностей. Побудовані моделі ілюструють логічну завершеність розробленого рішення та забезпечують основу для його практичної реалізації в наступному розділі.

РОЗДІЛ 4. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

4.1. Апаратні та програмні вимоги до системи

Розроблення та функціонування фреймворку автоматизованого тестування, створеного на основі Cypress та додаткових модулів, передбачає дотримання певних апаратних і програмних вимог, що забезпечують стабільність роботи, коректне виконання тестових сценаріїв, а також можливість одночасного запуску кількох компонентів системи. Враховуючи архітектуру рішення, яке включає браузерне середовище, Node.js сервер, модуль генерації тестів та інтерфейс користувача, важливо забезпечити такі умови, за яких усі зазначені елементи можуть працювати одночасно, без затримок, зависань або помилок, пов’язаних із нестачею ресурсів.

Для коректної роботи системи застосовувалися персональні комп’ютери середнього рівня продуктивності, що відповідають характерним вимогам сучасних інструментів автоматизації тестування (табл. 4.1).

Таблиця 4.1 – Апаратні вимоги для використання програми

Параметр	Мінімальні вимоги	Рекомендовані вимоги
Процесор	Intel Core i3 / AMD аналог	Intel Core i5 / Ryzen 5 або вище
Оперативна пам’ять	8 ГБ	16 ГБ і більше
Накопичувач	10 ГБ вільного простору	SSD 20+ ГБ вільного простору
Тип диска	HDD	SSD (для швидкого запуску Cypress та установки npm-пакетів)
Інтернет-з'єднання	стабільне, від 5 Мбіт/с	стабільне, від 20 Мбіт/с (для AI API)
Відеокарта	інтегрована	інтегрована / дискретна (не критично)

Основним параметром, який впливає на комфорт роботи, є обсяг оперативної пам’яті, оскільки одночасний запуск браузера, тестового раннера Cypress, а також Node.js сервера створює сумарне навантаження, яке менші конфігурації можуть

обробляти з затримками. Практичні експерименти під час розроблення фреймворку показали, що оптимальним є обсяг оперативної пам’яті не менше 8 ГБ, хоча для інтенсивної роботи з великими наборами тестів або для одночасного запуску кількох інстансів Cypress рекомендовано мати 16 ГБ або більше. Це дає можливість працювати без падіння продуктивності та забезпечує швидке перемикання між браузерними вкладками, редактором коду та середовищем запуску тестів.

Процесор також відіграє суттєву роль у загальній швидкості виконання тестів. Оскільки Cypress працює у браузері та виконує реальні операції над DOM, швидкість обробки подій визначається можливостями ядра процесора. Найбільш оптимальними для таких задач є процесори рівня Intel Core i5 або AMD Ryzen 5 і вище. Вони забезпечують достатню швидкодію для синхронного виконання тестових команд, рендерингу DOM і роботи JavaScript-движка. Для AI-модуля генерації тестів важливо мати стабільну роботу Node.js, однак даний модуль не створює значного навантаження на процесор, окрім короткочасного пікого навантаження в момент звернення до моделі.

Дисковий простір є необхідним для встановлення залежностей проєкту, таких як бібліотеки Node.js, Cypress та його власні компоненти, а також для збереження логів, тимчасових файлів, скриншотів і HTML-звітів. Cypress створює окремі каталоги для відеозаписів та скріншотів, тому рекомендовано мати не менше 10–15 ГБ вільного простору, щоб забезпечити належну роботу системи та можливість зберігати результати попередніх запусків. Для комфортної розробки застосовувався SSD-накопичувач, який значно пришвидшує встановлення залежностей, запуск Cypress, а також генерацію репортів.

Програмні вимоги системи безпосередньо пов’язані з технологічним стеком, використаним під час розроблення проєкту (табл. 4.2). Основою серверної частини є Node.js, тому необхідно мати встановлену стабільну версію Node.js (не нижче 18.x), оскільки вона забезпечує коректну роботу сучасних JavaScript-пакетів та модулів, включаючи Express, dotenv, OpenAI SDK та інші елементи, що складають інфраструктуру фреймворку. Менеджер пакетів npm використовується для встановлення залежностей і повинен бути оновленим до актуальної версії, щоб уникнути конфліктів під час збирання проєкту.

Таблиця 4.2. Програмні вимоги для використання програми

Компонент	Необхідна версія	Призначення
Операційна система	Windows 10 / Windows 11	Середовище розробки
Node.js	18.x або новіше	сервер AI-модуля, Cypress runner
npm	9.x або новіше	керування залежностями
Cypress	13.x або новіше	виконання автоматизованих UI-тестів
Браузер	Chrome, Edge або Firefox (остання версія)	середовище виконання тестів
Express.js	остання стабільна версія	запуск локального API сервера
dotenv	остання версія	робота з конфігурацією та API-ключами
OpenAI SDK	актуальна версія	генерація тестів за текстовим сценарієм
HTML/CSS UI	сучасні браузери	графічний інтерфейс для створення тестів

Для самої автоматизації застосовується Cypress версії 13 або вище. Ця версія має сучасні можливості тестування, актуальні синтаксичні зміни та стабільність, необхідну для роботи з більшістю вебзастосунків. Cypress працює з сучасними браузерами, такими як Google Chrome, Microsoft Edge та Mozilla Firefox, тому для коректного виконання тестів важливо мати встановлений хоча б один із них у актуальній версії. Це гарантує, що DOM API, JavaScript та інші елементи браузерної взаємодії працюватимуть належним чином.

Окрім серверної та браузерної частини, для роботи UI-модуля необхідно мати можливість запуску локального вебсервера, який надає доступ до HTML-інтерфейсу генерації тестів. Внутрішня панель працює через Express та може бути доступною будь-якому браузеру. Також система використовує OpenAI API-key, який зберігається у файлі конфігурації .env відповідно до вимог безпеки.

Таким чином, апаратні та програмні вимоги, описані вище, формують базу, що забезпечує стабільну та передбачувану роботу фреймворку автоматизованого тестування. Дотримання цих вимог гарантує швидке виконання тестових сценаріїв, коректну роботу AI-модуля та можливість отримувати детальні звіти про тестування без втрати продуктивності.

4.2. Архітектура, структура та інтегровані компоненти розробленого фреймворку

Розроблений фреймворк автоматизованого тестування являє собою багаторівневу систему, у якій усі компоненти тісно інтегровані між собою та працюють відповідно до єдиної логічної моделі. Архітектура фреймворку побудована так, щоб забезпечити максимальну модульність, розширюваність і зручність використання, а також можливість інтеграції з сучасними інструментами штучного інтелекту, які генерують тестові сценарії на основі текстового опису. Центральним елементом системи виступає Cypress –фреймворк для енд-ту-енд тестування, що забезпечує безпосередню взаємодію з вебінтерфейсом у режимі реального часу. Усі інші компоненти –сервер генерації тестів, графічний інтерфейс, модуль Page Object Model, система звітності, а також допоміжні скрипти та модулі обробки –побудовані навколо Cypress та взаємодіють із ним як зі стрижневим механізмом виконання тестів.

Архітектура проєкту (рис. 4.1) побудована на основі клієнт-серверної моделі, де UI виконує функцію зовнішнього шару, а бекенд на базі Node.js забезпечує логіку генерації тестів та службову взаємодію між компонентами. Всі тестові сценарії зберігаються у структурованих директоріях Cypress, що полегшує їх підтримку та систематизацію. Зокрема, фреймворк містить окремі папки для згенерованих AI-сценаріїв, вручну написаних тестів, Page Object моделей, фікстур, службових файлів та конфігурацій. Така структуризація є важливою з точки зору масштабованості проєкту, оскільки дозволяє без труднощів додавати нові модулі або розширювати функціональність на наступних етапах.

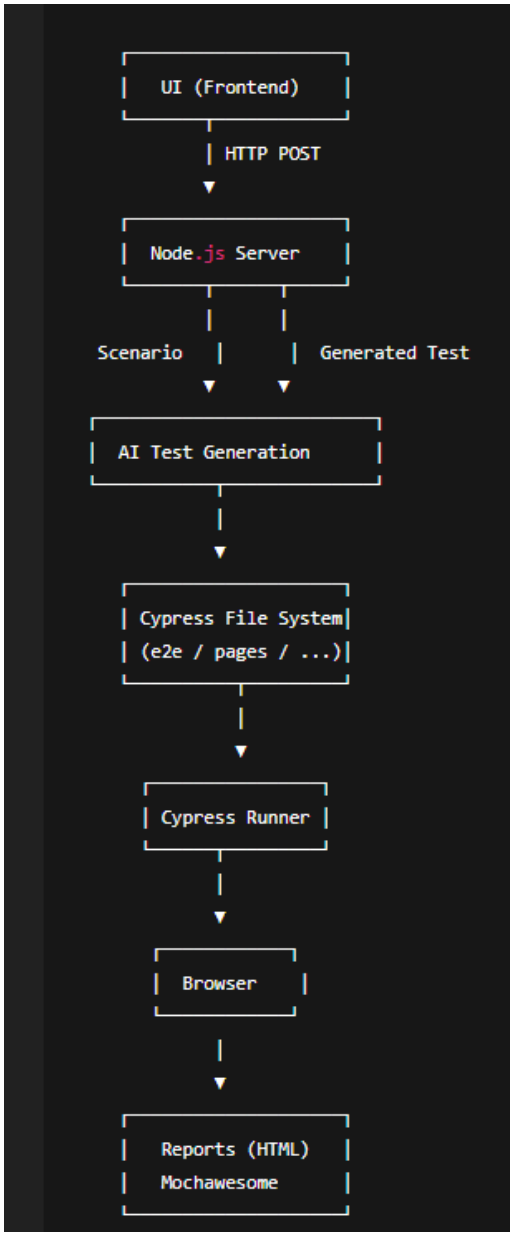

Рисунок 4.1 – Архітектура проєкту

У процесі розроблення фреймворку першочерговим завданням стало створення базових автоматизованих тестів на основі Cypress, що дозволило сформувати фундаментальне розуміння структури вебзастосунку, визначити взаємозв’язки між його сторінками та підготувати основу для подальшої масштабної автоматизації. На цьому етапі були розроблені тести для найважливіших бізнес-сценаріїв, включаючи авторизацію користувача, відображення та взаємодію з продуктами, перевірку функціональності сортування та повний цикл оформлення покупки. Створення цих тестів вручну дало змогу оцінити стабільність інтерфейсу, визначити критичні елементи, що вимагають автоматизації, і вибудувати структуру, яка згодом стала невід’ємною частиною архітектури фреймворку.

Під час написання тестів використовувалася архітектура Page Object Model, що забезпечила чіткий поділ логіки взаємодії з інтерфейсом від самих тестових сценаріїв. На прикладі тестів авторизації (рис. 4.2.) можна побачити, що окремі методи сторінки логіну відповідають за перехід на сторінку, заповнення поля імені користувача, введення пароля та натискання кнопки входу. Такий підхід уможливив побудову не лише позитивних сценаріїв, а й негативних, таких як логін із заблокованим або проблемним користувачем. Це забезпечило високу читабельність тестів (рис. 4.3) і значно спростило їх підтримку.

```
class LoginPage {
    visit() {
      cy.visit('/')
    }

    fillUsername(name) {
      cy.get('[data-test="username"]').type(name)
    }

    fillPassword(password) {
      cy.get('[data-test="password"]').type(password)
    }

    submit() {
      cy.get('[data-test="login-button"]').click()
    }

    loginAs(user, pass) {
      this.visit()
      this.fillUsername(user)
      this.fillPassword(pass)
      this.submit()
    }
    error() {
    return cy.get('[data-test="error"]')
  }
  }

export default new LoginPage()
```

Рисунок 4.2 – Приклад Page Object Model для сторінки авторизації

```
import LoginPage from '../pages/LoginPage'

describe('Login Tests', () => {
  beforeEach(() => {
    cy.fixture('users').as('users')
  })

  it('Valid login', { tags: ['@smoke','@critical'] }, function () {
    LoginPage.loginAs(this.users.validUser.username, this.users.validUser.password)
    cy.url().should('include', '/inventory.html')
  })

  it('Locked out user', { tags: ['@regression','@critical'] }, function () {
    LoginPage.loginAs(this.users.lockedUser.username, this.users.lockedUser.password)
    cy.get('[data-test="error"]').should('contain', 'locked out')
  })

  it('Problem user login', { tags: ['@regression','@critical'] }, function () {
    LoginPage.loginAs(this.users.problemUser.username, this.users.problemUser.password)
    cy.url().should('include', '/inventory.html')
  })
```

Рисунок 4.3 – Приклад авто-тестів для авторизації

Аналогічним чином були описані й інші сторінки застосунку. Для переліку продуктів створено методи, що відповідають за відображення товарів, взаємодію з кнопками додавання до кошика та перевірку коректності сортування. Окремий Page Object був створений для кошика, де реалізовано логіку перевірки доданих товарів (рис. 4.4), переходу до оформлення замовлення та заповнення обов'язкових полів (рис 4.5).

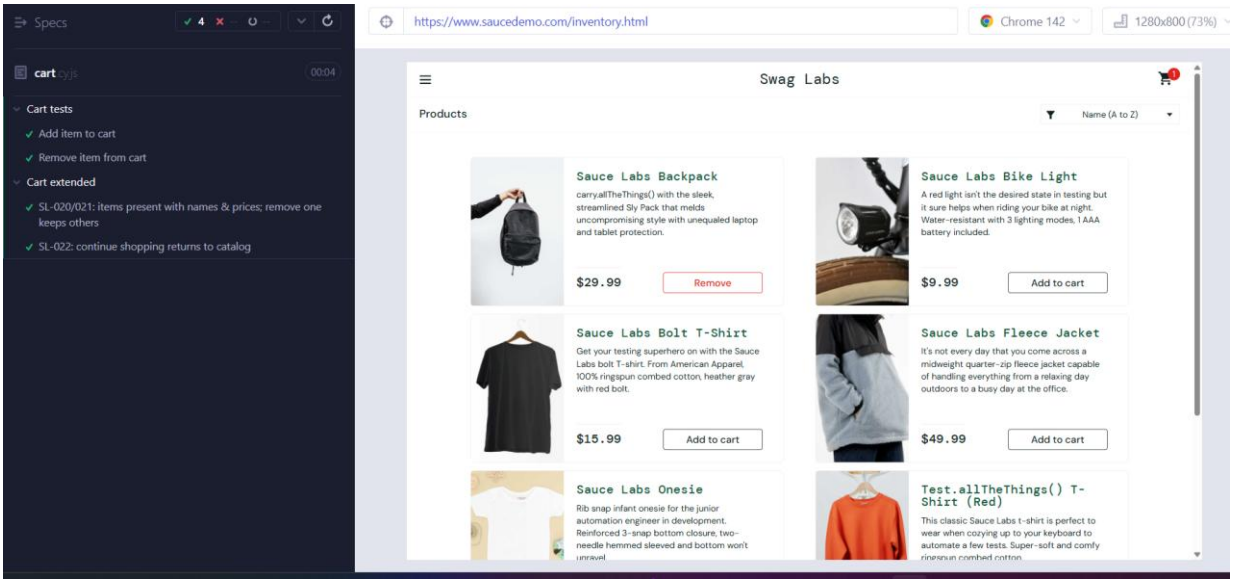

Рисунок 4.4 – Реалізація перевірки доданих товарів

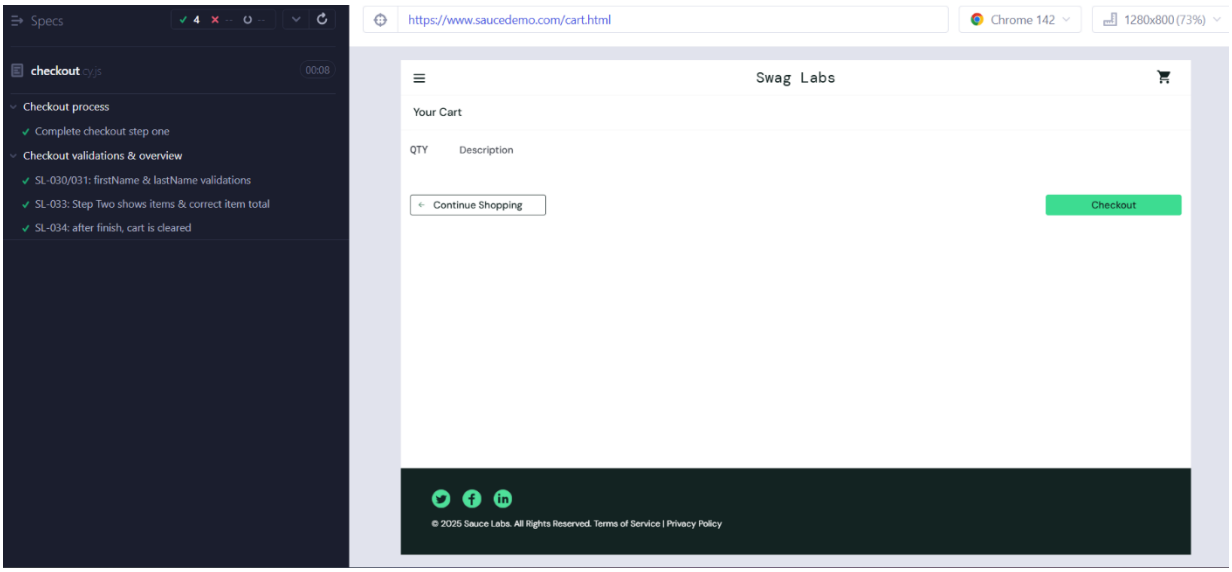

Рисунок 4.5 – Реалізація тестів оформлення замовлення та заповнення обов'язкових полів

Використання фікстур (рис. 4.6), зокрема JSON-файлу з даними для оформлення замовлення, дало змогу відокремити тестові дані від логіки сценаріїв і забезпечило можливість повторного використання цих даних у різних тестах. У результаті була сформована стійка, масштабована структура, що поклала початок розробленню повноцінного фреймворку.

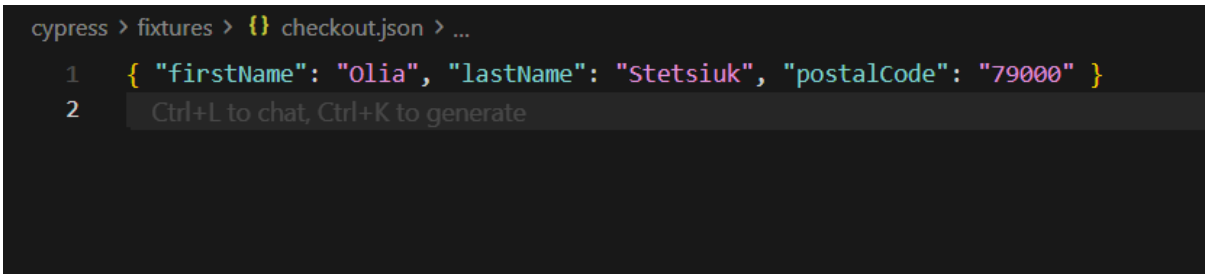

Рисунок 4.6 – Приклад фікстури

На наступному етапі в систему було інтегровано AI-модуль автоматичної генерації тестових сценаріїв (рис. 4.7). Його роль полягає у перетворенні текстового опису сценарію, який користувач вводить у графічному інтерфейсі, у повноцінний Cypress-тест.

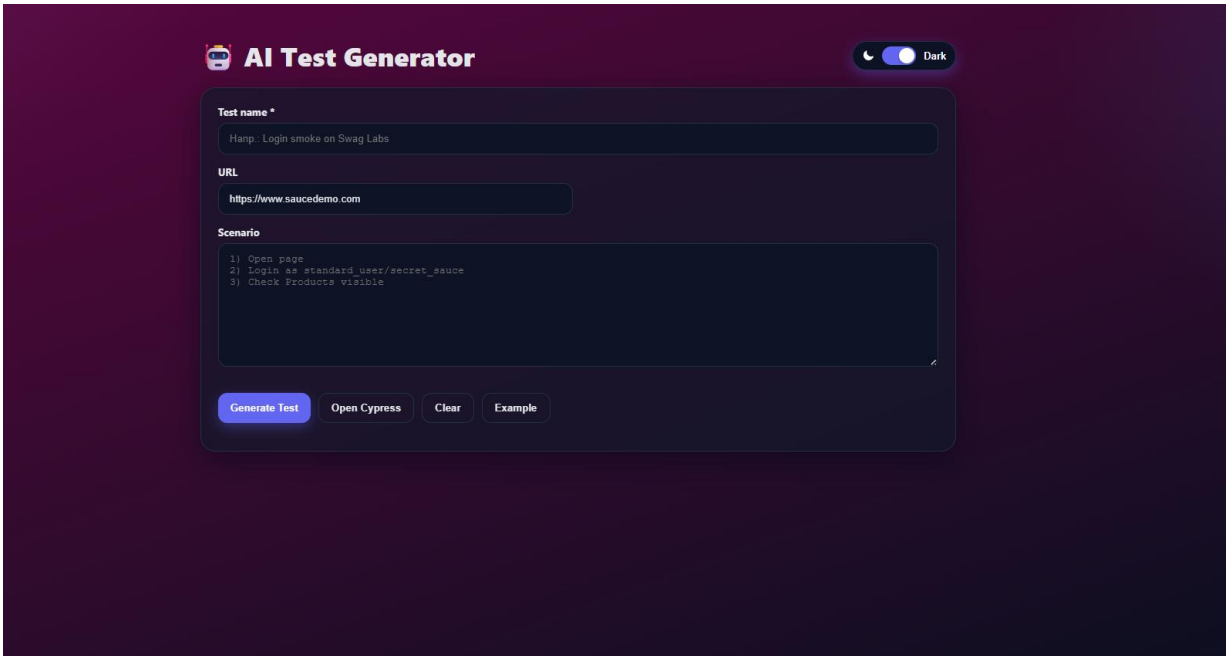

Рисунок 4.7 – Вигляд AI-модуль автоматичної генерації тестових сценаріїв

Модуль приймає від UI текст сценарію та надсилає його на Node.js сервер, де відбувається обробка. Сервер передає отриманий текст у AI-компонент, що виконує семантичний аналіз і визначає логічну структуру дій, яку має виконати тест (рис.4.8). На основі цього аналізу генерується Cypress-код, що містить всю логіку сценарію: відкриття сторінки, взаємодію з елементами та перевірку очікуваних результатів.

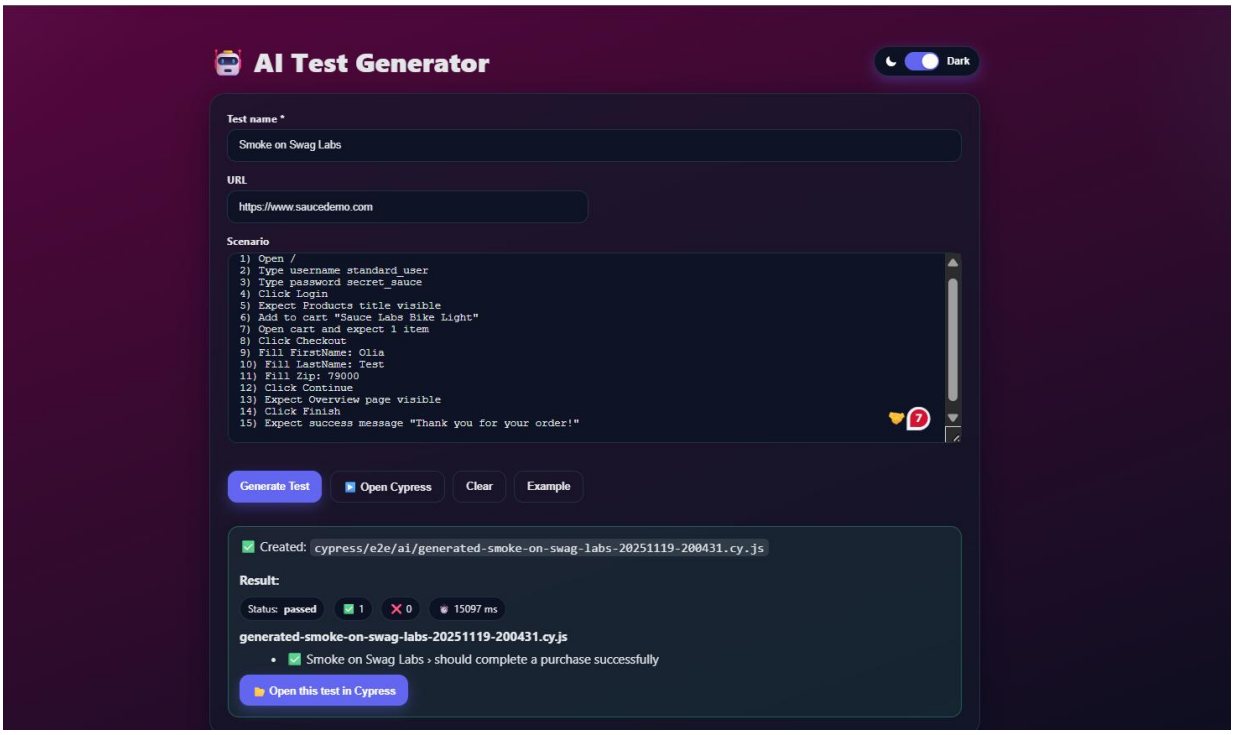

Рисунок 4.8 – Приклад генерації сценарію і його виконання

AI-модуль працює не ізольовано, а безпосередньо взаємодіє зі структурою Page Object Model, що була створена на попередньому етапі. Завдяки цьому згенеровані тести не дублюють локатори чи базові команди Cypress, а використовують готові методи класів сторінок. Наприклад, якщо у тексті сценарію міститься інструкція щодо авторизації, модуль автоматично формує виклик методу loginAs(), який виконує весь процес входу. Це забезпечує узгодженість між ручними та автоматично згенерованими тестами й робить фреймворк єдиною структурною системою. Сформований тестовий файл зберігається у відповідній директорії, де одразу стає доступним для перегляду та запуску.

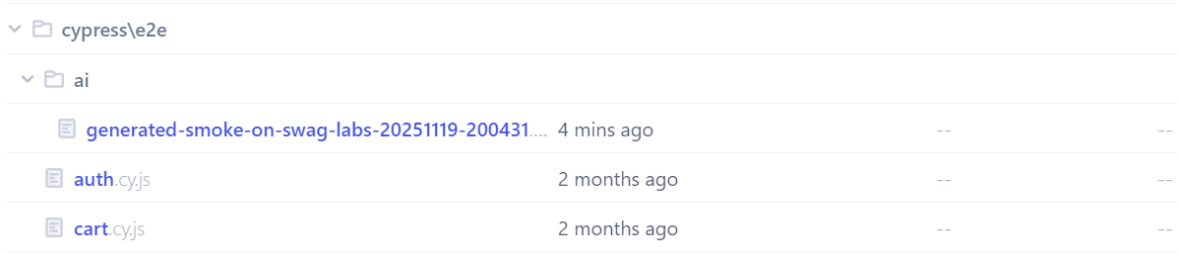

Рисунок 4.9 – Приклад збереженого згенерованого тесту

Завершальним компонентом архітектури фреймворку є система логування та звітності, яка фіксує результати кожного запуску тестів. Після завершення виконання сценарію Cypress автоматично генерує структурований JSON-файл, у якому міститься повна статистика тестування: кількість тестів, статус їх проходження, відсоток успішних перевірок, час виконання та загальна тривалість. На основі цих даних формується фінальний HTML-звіт за допомогою репортера Mochawesome, що представляє результати у зручному графічному вигляд (рис. 4.10).

Рисунок 4.10 – Приклад Mochawesome-логу

Крім того, система створює відеозапис (рис. 4.11) проходження тесту, який може бути використаний для аналізу нестабільних поведінкових сценаріїв або для демонстрації результатів команді. Завдяки таким артефактам фреймворк забезпечує стабільний, прозорий і відтворюваний процес тестування, незалежно від того, був тест створений вручну чи згенерований за допомогою AI-модуля.

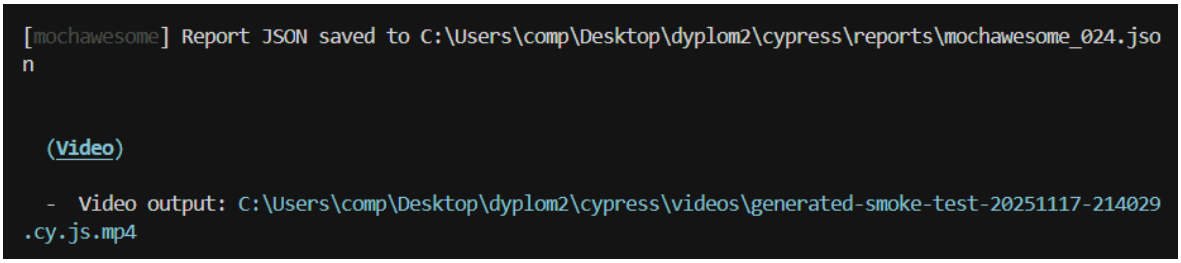

Рисунок 4.11 – Приклад відеозапису виконаного тесту

4.3. Оцінка ефективності розробленого фреймворку

Ефективність розробленого фреймворку оцінювалася шляхом порівняння ручного процесу створення автоматизованих тестів із процесом їх автоматизованої генерації за допомогою AI-модуля. Під час аналізу враховувалися такі параметри, як швидкість створення тестових сценаріїв, стабільність виконання, відтворюваність результатів, а також обсяг та інформативність технічних артефактів, сформованих під час запуску тестів. Такий підхід дав змогу об'єктивно оцінити практичну цінність інтеграції AI-технологій у класичний Cypress-фреймворк та визначити рівень оптимізації, досягнутий завдяки впровадженню комплексного рішення.

У ході дослідження було виміряно час, необхідний для створення базового smoke-тесту, що включає відкриття сторінки, авторизацію, перехід на сторінку товарів, взаємодію з продуктом та перевірку коректності навігації. Ручне написання аналогічного тесту займало від 12 до 20 хвилин залежно від складності локаторів і кількості повторних запусків, необхідних для коригування сценарію. Натомість створення такого самого тесту за допомогою AI-модуля тривало в середньому 2–4 секунди. У результаті продуктивність процесу зросла приблизно в 170–300 разів, що демонструє суттєву перевагу автоматичної генерації над традиційним ручним написанням.

Окрім суттєвого пришвидшення, важливим показником ефективності стала стабільність виконання згенерованих тестів. Завдяки вбудованим механізмам синхронізації Cypress тестові сценарії виконувалися у передбачуваній послідовності незалежно від можливої різниці у швидкості завантаження сторінок або короткочасних затримок браузера. На прикладі одного з успішно згенерованих smoke-тестів видно, що тест завершився зі статусом passed за 15097 мілісекунд без необхідності ручних виправлень чи адаптації коду. Це підтверджує коректність роботи AI-модуля, який не лише формує валідний синтаксис, а й створює логічно завершені та стабільні сценарії.

Високу інформативність тестового процесу забезпечила система звітності, інтегрована у фреймворк. Після кожного запуску система формувала JSON-лог із детальною статистикою, HTML-звіт через Mochawesome, а також відеозапис

проходження сценарію. Наявність цих артефактів дозволила верифікувати правильність роботи фреймворку, аналізувати часові характеристики, контролювати якість виконання кроків і швидко виявляти потенційні проблеми. Така структурованість забезпечує прозорість процесу тестування та спрощує його підтримку в довгостроковій перспективі.

Значущою перевагою фреймворку є те, що AI-модуль працює у повній відповідності до створеної структури Page Object Model, не дублює локатори та слабко пов’язані команди, а використовує вже існуючу архітектуру. Це суттєво підвищує якість та однорідність згенерованих тестів і значно спрощує їх подальший супровід, незалежно від того, були вони створені вручну чи автоматично.

Підсумовуючи, проведена оцінка продемонструвала, що розроблений фреймворк забезпечує істотне скорочення часу на створення автоматизованих тестів, високий рівень стабільності, передбачуваності та відтворюваності виконання. Поєднання класичного інструментарію Cypress з можливостями штучного інтелекту дозволило створити гнучку, масштабовану та продуктивну систему, яка ефективна у реалізації реальних проєктів із великим обсягом тестових сценаріїв.

Висновок до розділу

У цьому розділі було представлено повний цикл розроблення фреймворку автоматизованого тестування, що поєднує класичні підходи Cypress із сучасними методами генерації тестових сценаріїв на основі штучного інтелекту. Виконана робота показала, що початкове створення ручних автотестів дозволило сформувати чітку архітектуру Page Object Model, визначити ключові сторінки застосунку, описати логіку взаємодії з елементами інтерфейсу та підготувати стабільні шаблони для подальшої автоматизації. Побудована структура стала фундаментом, на який було інтегровано AI-модуль, що забезпечує автоматичне створення тестів на основі текстових інструкцій користувача.

Розроблений AI-модуль суттєво розширив можливості фреймворку, перетворивши процес створення тестових сценаріїв на швидку та більш доступну операцію. Інтеграція з Node.js сервером та використання вже існуючої архітектури

Page Object Model дозволили автоматично генерувати структурований та читабельний Cypress-код, який відповідає вимогам системи й не потребує ручного доопрацювання. Це дало можливість формувати нові тести за лічені секунди, що значно підвищило ефективність роботи.

Окрему роль у формуванні повного циклу тестування відіграла система логування та звітності. Створення JSON-файлів, HTML-звітів та відеозаписів виконання тестів забезпечило прозорість, відтворюваність та можливість детального аналізу результатів. Завдяки використанню Mochawesome фреймворк отримав механізм, який дозволяє документувати проходження тестів і швидко виявляти можливі відхилення у поведінці сценаріїв.

Узагальнюючи, розроблений фреймворк є комплексним рішенням, що об’єднує ручні та автоматично генеровані тести, інструменти сучасної автоматизації та можливості штучного інтелекту. Такий підхід забезпечує масштабованість, стабільність і значне прискорення процесу створення тестових сценаріїв, що робить систему ефективною у реальних умовах використання та придатною для подальшого розширення.

РОЗДІЛ 5. РОЗРОБЛЕННЯ СТАРТАП-ПРОЄКТУ

5.1. Опис ідеї проєкту

Ідея стартап-проєкту полягає у створенні сучасної платформи, яка поєднує можливості автоматизованого тестування вебзастосунків із технологіями штучного інтелекту та дозволяє значно спростити процес формування end-to-end тестів. Основою системи виступає інтелектуальний модуль, здатний перетворювати текстовий опис дій користувача, поданий природною мовою, у повноцінний та готовий до виконання тестовий сценарій для фреймворку Cypress. Це означає, що спеціалісту більше не потрібно писати код вручну, продумувати структуру, працювати з локаторами або підтримувати власну тестову інфраструктуру – достатньо сформулювати послідовність дій у звичній текстовій формі, після чого система самостійно сформує коректний тестовий файл, збереже його в проєкті й забезпечить можливість подальшого запуску.

Суть проєкту полягає у вирішенні однієї з ключових проблем сучасного тестування, коли створення автоматизованих тестів вимагає значних часових витрат, технічних компетенцій та постійної підтримки. Більшість команд стикаються з тим, що, попри бажання автоматизувати процеси, вони не мають можливості забезпечити швидке і якісне написання тестових сценаріїв. Це стосується як стартапів, які не мають окремих автоматизаторів, так і великих компаній, де об’єм тестів постійно зростає, а створення кожного нового сценарію займає значний час. Запропонована платформа усуває ці обмеження, перетворюючи текстові сценарії у повноцінний код і тим самим робить автоматизацію доступною навіть для початківців або для команд, що працюють у швидкому циклі розроблення.

Ідея проєкту охоплює не лише механізм генерації тестів, а й створення цілісної екосистеми, яка дозволяє користувачу одразу запускати створені тести та переглядати результати у зрозумілому та структурованому вигляді. Система об’єднує UI для введення сценаріїв, серверну частину, яка відповідає за генерацію та збереження коду, і інтеграцію з Cypress, що забезпечує автоматичний запуск сценаріїв та формування звітів. Таким чином, користувач отримує повний цикл

роботи над тестом: від формування сценарію до отримання JSON-логів, HTML-звітів і відеозаписів проходження тестів. Це робить продукт придатним не лише для створення тестів, але й для проведення регулярного контролю якості, використання у CI/CD-процесах та аналітики результатів тестування [2].

Розроблена платформа має широкий спектр застосування, адже вона може використовуватися як окремими тестувальниками, так і командами будь-якого розміру. Її можна застосовувати для генерації smoke-, regression- або functional-тестів, а також для швидкої автоматизації критичних бізнес-процесів. Особливо ефективним застосування платформи є в умовах інтенсивної розробки, коли продукт оновлюється щоденно, а командам потрібно максимально швидко формувати нові сценарії або оновлювати існуючі. Наявність можливості створення тестів природною мовою багаторазово знижує навантаження на інженерів з автоматизації та розширює можливості ручних тестувальників, які до цього не мали технічних навичок написання коду.

Однією з ключових переваг ідеї є її унікальність у порівнянні з існуючими на ринку рішеннями. Попри наявність платформ, які застосовують елементи штучного інтелекту для автоматизації тестування, більшість із них мають низку обмежень: закриті екосистеми, високі цінові бар’єри, використання власних синтаксичних форматів замість Cypress, відсутність можливості гнучкого редагування тестів або складність інтеграції з CI/CD. На відміну від цих продуктів, запропонований стартап орієнтований саме на генерацію чистого, зрозумілого й редагованого Cypress-коду, що робить його легко інтегрованим у будь-який існуючий проєкт. При цьому платформа зберігає простоту використання та не вимагає спеціальних знань у сфері програмування, що дозволяє залучати до автоматизації ширшу аудиторію.

У цілому ідея проєкту спрямована на створення доступного, технологічно гнучкого та масштабованого інструмента, що перетворює підхід до автоматизації тестування. Платформа не лише спрощує процес створення тестів, але й дозволяє будувати стабільну, ефективну та високопродуктивну систему контролю якості, здатну задовольнити потреби сучасного ринку та адаптуватися до зростаючих вимог користувачів і компаній.

5.2. Аналіз технологічних можливостей реалізації ідей проєкту

Реалізація AI-платформи для автоматичної генерації Cypress-тестів базується на сучасних вебтехнологіях, які вже доступні на ринку та мають високу стабільність, підтримку спільноти й великий потенціал масштабування. Технологічна концепція проєкту не потребує створення нових унікальних мов програмування або спеціалізованих інструментів, а спирається на комбінацію перевірених рішень, що забезпечують високу надійність системи, швидкість виконання операцій та можливість інтеграції з будь-яким існуючим IT-середовищем. Основою проєкту виступають три ключові технологічні платформи: Node.js як середовище виконання серверної частини, Cypress як фреймворк автоматизованого тестування та сучасні моделі штучного інтелекту, здатні аналізувати природну мову й перетворювати текст на структуровані інструкції.

З технологічної точки зору проєкт будується на модульній архітектурі, де кожен компонент виконує чітко визначену функцію і при цьому легко замінюється або оновлюється без шкоди для роботи всієї системи. Серверна частина реалізується за допомогою Node.js, що забезпечує високу продуктивність та можливість обробки великої кількості запитів у режимі реального часу. Це дозволяє користувачу працювати з платформою у вебінтерфейсі: вводити сценарій, надсилати запит, отримувати сформований файл тесту та запускати його практично миттєво. Node.js також дозволяє інтегрувати інструменти для запуску Cypress у фоновому режимі, завдяки чому весь цикл –від введення тексту до перегляду результатів тестування – здійснюється в межах однієї системи.

Іншим ключовим елементом реалізації є Cypress –сучасний фреймворк для автоматизації тестування вебінтерфейсів, який відрізняється швидкістю виконання тестів, вбудованою синхронізацією команд і зручними механізмами взаємодії з DOM. Cypress повністю підтримує JavaScript, що дозволяє платформі згенерувати тест у вигляді чистого вихідного коду, який може бути запущений безпосередньо, без проміжного перетворення або компіляції. Саме ця властивість Cypress створює можливість легко вбудовувати сгенеровані AI-тести у процеси CI/CD, а також виконувати їх у локальному середовищі. Додатковою перевагою Cypress є розвинена

система звітності – JSON-логи, HTML-репорти, відеозаписи проходжень тестів – що дозволяє робити аналіз результатів максимально прозорим та технічно обґрунтованим.

Особливу роль у реалізації проєкту відіграє модуль штучного інтелекту, який аналізує текстові сценарії, формує структуровані інструкції та генерує код. Технологічно система спирається на сучасні моделі обробки природної мови, здатні розуміти послідовність дій, контекст і логіку користувацького сценарію. Завдяки цьому платформа не прив'язана до шаблонних команд, а може обробляти складні або нестандартні інструкції, визначаючи логічну структуру тесту та формуючи відповідний JavaScript-код. Такий підхід дозволяє забезпечити гнучкість і адаптивність системи, а також масштабувати її можливості під різні типи вебзастосунків – від простих односторінкових сайтів до складних корпоративних систем.

Технологічна здійсненність проєкту підтверджується тим, що всі ключові компоненти – Node.js, Cypress та AI-моделі – є доступними для використання, не потребують ліцензування або дорогих платних сервісів (окрім опціонального використання комерційних API), а також мають великий обсяг документації та спільнотної підтримки. Це забезпечує можливість швидкої розробки, зручного оновлення та подальшого розширення функціональності. Крім того, інструменти є кросплатформеними, що дозволяє запускати платформу на Windows, Linux або macOS без необхідності ускладненого налаштування або адаптації коду.

Аналіз технологічної бази показує, що реалізація проєкту є не лише можливою, але й оптимальною в умовах сучасного розвитку автоматизованого тестування. Наявність перевірених технологій, можливість інтеграції з CI/CD та простота масштабування системи роблять стартап технічно здійсненним з мінімальними ризиками. Вибір стеку також відповідає тенденціям ринку, де переважає використання JavaScript, хмарних сервісів та автоматизованих інструментів аналізу. За результатами аналізу можна зробити висновок, що запропонована ідея є повністю технологічно здійсненною, а її реалізація може бути виконана за допомогою доступних, стабільних та сучасних технологій, що гарантують надійність і продуктивність майбутнього продукту.

5.3. Аналіз ринкових можливостей запуску стартап-проєкту

Ринок автоматизованого тестування програмного забезпечення вже протягом кількох років демонструє стабільне зростання, що зумовлено переходом більшості компаній на прискорені моделі розроблення, такі як Agile та DevOps. Швидкі релізи, часті оновлення функціональності та необхідність забезпечення високої якості продукту створюють потребу у масштабуванні тестових процесів, які вже неможливо підтримувати виключно ручними методами. Саме тому інструменти автоматизації стали ключовим елементом більшості технологічних компаній, а попит на ефективні рішення у сфері тестування постійно збільшується. На цьому фоні поява платформи, здатної автоматично генерувати тести за допомогою штучного інтелекту, відкриває значні ринкові можливості для впровадження стартап-проєкту.

Сучасний ринок QA та automation QA характеризується кількома важливими тенденціями, які безпосередньо впливають на перспективність пропонованого продукту. По-перше, зростає кількість компаній, що переходять до CI/CD-підходів, а це вимагає створення великої кількості автоматизованих тестів, здатних запускатися на кожному етапі розгортання. Багато команд стикаються з тим, що написання та підтримка автоматизованих тестів забирає значну частину часу й ресурсу, що може гальмувати процес розроблення. Платформа, яка дозволяє швидко створювати сценарії без необхідності писати код, здатна повністю змінити підхід до автоматизації та значно зменшити завантаження QA-команд [17].

Дослідження ринку показує, що компанії середнього та малого сегменту особливо гостро потребують інструментів, які дають можливість отримувати переваги автоматизації без залучення висококваліфікованих спеціалістів. Більшість стартапів та продуктових компаній не мають окремих automation інженерів і покладаються на ручних тестувальників, які часто не можуть дозволити собі поглиблене навчання мовам програмування або складним фреймворкам. Завдяки простоті взаємодії з платформою, де користувачу потрібно лише сформулювати тест у текстовій формі, продукт отримує значну конкурентну перевагу та відкриває для себе широку аудиторію, яка сьогодні не охоплена інструментами автоматизації.

Важливо також враховувати загальну динаміку світового ринку програмних продуктів, які використовують штучний інтелект для оптимізації робочих процесів. Індустрія активно рухається в напрямі автоматизації рутинних завдань, а тестування є однією з найбільш трудомістких та повторюваних частин розроблення. Застосування штучного інтелекту у цій сфері дозволяє не лише зменшити кількість ручної роботи, але й підвищити рівень стандартизації та зменшити кількість людських помилок. На цьому фоні стартап-проєкт вписується в глобальну тенденцію й отримує підтримку з боку ринку, який вже готовий до активного впровадження подібних рішень.

Поряд із ринковими можливостями існують і певні загрози, які потрібно врахувати. Конкуренція у сфері інструментів тестування є доволі високою, оскільки на ринку вже присутні такі продукти, як Testim, Mabl, Functionize, а також рішення, що використовують AI-підхід у різних формах. Однак аналіз показує, що більшість конкурентів орієнтовані на корпоративний сектор і пропонують складні, багатофункціональні та дорогі продукти, які не завжди доступні для широкого кола користувачів. Пропонована платформа відрізняється простотою інтеграції, доступністю та можливістю працювати з Cypress –фреймворком, який займає провідну позицію на ринку автоматизації вебзастосунків. Це дозволяє зосередитися на сегменті, де попит на доступні та прості у використанні інструменти значно перевищує пропозицію.

Потенційними користувачами системи можуть бути компанії різних масштабів –від початківців, які лише формують процеси тестування, до великих підприємств, що прагнуть зменшити навантаження на існуючі automation-команди. У межах цього ринку формується кілька груп клієнтів, кожна з яких має власні потреби. Ручні тестувальники прагнуть отримати інструмент, який дозволяє автоматизувати частину роботи без необхідності вивчення складних технічних інструментів. Automation-інженери можуть використовувати платформу для швидкого формування шаблонів, генерації рутинних тестів або прискорення smoke-перевірок перед релізами. Продуктові команди можуть застосовувати платформу для швидкого формування базових перевірок у межах демонстрацій або тестування

прототипів. Таким чином, аудиторія стартапу є широкою і охоплює різні сегменти ринку.

Ринкове середовище сьогодні загалом сприятливе для впровадження продуктів, заснованих на AI-автоматизації. Користувачі активно шукають способи прискорити тестування, зменшити витрати й підвищити надійність продуктів. Це створює основу для запуску та масштабування запропонованої платформи, яка відповідає сучасним потребам галузі та має реальні конкурентні переваги. За результатами ринкового аналізу можна зробити висновок, що продукт має високий потенціал комерціалізації, здатний зайняти нішу серед інструментів, орієнтованих на доступність, швидкість і простоту використання.

5.4. Розроблення ринкової стратегії проєкту

Розроблення ринкової стратегії для стартап-проєкту штучного інтелекту з автоматичної генерації Cypress-тестів передбачає визначення цільових груп користувачів, принципів охоплення ринку та підходів до їх залучення з урахуванням специфіки сучасної індустрії тестування. Оскільки продукт базується на інноваційному рішенні, здатному змінити традиційний підхід до автоматизації, необхідно формувати стратегію таким чином, щоб максимально відобразити його унікальність, цінність і практичну користь для різних категорій користувачів.

У процесі формування ринкової стратегії важливо визначити, які групи користувачів є найбільш зацікавленими у впровадженні інструментів такого типу. З огляду на попередній аналіз ринку, можна зробити висновок, що основними споживачами є тестувальники різного рівня, команди автоматизації, продуктові та аутсорсингові компанії, стартапи, а також невеликі команди розробників, які прагнуть швидко підвищити якість продукту. Система, яка генерує тести природною мовою, відкриває можливість охопити широку аудиторію, включно з тими користувачами, які раніше не могли повноцінно користуватися автоматизацією через складність технічних інструментів. Таким чином, ринкова стратегія будується на принципі доступності, простоти та швидкості, що відповідає потребам усіх визначених сегментів.

З позиції стратегічного охоплення ринку найдоцільнішим є поєднання елементів концентрованого та диференційованого маркетингу. На першому етапі розвитку продукт фокусується на одному сегменті –інженерах з ручного тестування, які мають потребу в автоматизації, але не завжди володіють необхідними технічними навичками. Саме для цієї категорії користувачів продукт приносить найвищу додану вартість, оскільки дозволяє їм розширити професійні можливості та значно збільшити власну продуктивність. Паралельно зі зростанням аудиторії та стабілізацією продукту стратегія може переходити до диференційованого підходу, у якому пропозиція адаптується для інших сегментів –команд автоматизації, проєктів із високим рівнем складності, компаній, що використовують CI/CD, або великих організацій, які потребують масштабного автоматизованого тестування.

У межах стратегії важливо визначити також ціннісну пропозицію продукту, яка формує основу маркетингової комунікації. Унікальність платформи полягає у здатності значно скоротити час створення тестових сценаріїв, спростити процес автоматизації й забезпечити можливість швидко масштабувати тестове покриття без залучення додаткових ресурсів. Тому продукт позиціонується як рішення, здатне не лише підвищити ефективність тестування, але й забезпечити конкурентну перевагу для компаній, які прагнуть швидше виходити на ринок і підтримувати високу стабільність застосунків.

Стратегія поширення продукту повинна враховувати канали комунікації, що найбільш ефективно взаємодіють з цільовою аудиторією. Оскільки більшість потенційних користувачів працюють у галузі інформаційних технологій, важливу роль відіграє цифровий маркетинг: технічні форуми, професійні спільноти, соціальні мережі, вебінари, демонстраційні відео та навчальні матеріали. Особливу увагу варто приділити співпраці з QA-спільнотами, участі в конференціях та онлайн-подіях, що дозволяє демонструвати практичні можливості продукту й залучати зацікавлену аудиторію.

Додатково стратегія повинна враховувати можливість партнерств із навчальними закладами, технічними школами та компаніями, що спеціалізуються на підготовці тестувальників. Інтеграція продукту в освітні програми дозволяє

створити базу користувачів ще на етапі їх професійного розвитку, що формує стабільний попит у майбутньому та сприяє ширшому розповсюдженню рішення.

Формування ринкової стратегії також включає визначення підходів до монетизації, які забезпечують фінансову стійкість стартапу. Найбільш доцільною у цьому випадку є модель підписки, яка передбачає щомісячні або щорічні платежі в залежності від обсягу використання продукту. Такий підхід дозволяє побудувати прогнозований і стабільний дохід, водночас залишаючи можливість впровадження гнучких тарифів для різних груп користувачів. Для невеликих команд може бути створений базовий тариф із обмеженою кількістю тестів, для середніх та великих компаній –розширені тарифи з доступом до інтеграцій, API, командної роботи та історії запусків.

Отже, ринкова стратегія стартапу передбачає поступовий і продуманий вихід на ринок, де продукт позиціонується як інноваційне рішення, доступне для широкої аудиторії, здатне задовольнити потреби як новачків, так і досвідчених інженерів. Такий підхід дозволяє сформувати сильну та конкурентоспроможну позицію продукта на ринку та забезпечити стабільний розвиток у середньо- та довгостроковій перспективі.

5.5. Маркетингова програма стартап-проєкту

Маркетингова програма стартап-проєкту повинна забезпечувати ефективне позиціонування продукту на ринку, формування попиту та створення стійкої системи взаємодії з потенційними користувачами. Оскільки продукт є інноваційним та орієнтованим на технологічну аудиторію, маркетингова стратегія повинна враховувати специфіку поведінки споживачів у сфері інформаційних технологій, їх цифрову активність, залежність від професійних спільнот та високу чутливість до якості інструментів. Головним завданням маркетингової програми є формування іміджу платформи як корисного, доступного та високотехнологічного інструмента, що допомагає суттєво скоротити витрати часу та ресурсів на тестування.

Просування стартапу повинно базуватися на демонстрації реальної цінності продукту, адже у сфері тестування найбільш переконливими є не рекламні

матеріали, а конкретні результати. Саме тому важливим елементом маркетингу є публічна демонстрація функціональності системи: створення відео з генерацією тестів у реальному часі, тестових запусків, прикладів звітів, порівняння швидкості ручного та автоматичного створення сценаріїв. Такий підхід формує довіру аудиторії й дає змогу потенційним клієнтам побачити практичне застосування інструмента та його ефективність у реальних умовах.

З огляду на те, що більшість представників цільової аудиторії активно користуються професійними спільнотами, соціальними мережами та технічними форумами, важливу роль у маркетинговій програмі відіграватимуть онлайн-канали. Платформа має бути присутньою в середовищах, де розробники та тестувальники обмінюються досвідом: GitHub, LinkedIn, Stack Overflow, YouTube, а також у численних QA-спільнотах. Публікація інструкцій, демонстрацій, покрокових гайдів і реальних кейсів використання дозволить сформувати перше ядро зацікавлених користувачів та почати органічне зростання попиту. Одним із ключових напрямів стане створення навчальних матеріалів: простих відеокурсів, статей про автоматизацію тестування та вебінарів для новачків, які дадуть їм можливість безболісно перейти з ручного тестування до автоматизованого.

Маркетингова програма повинна також включати співпрацю з освітніми платформами та курсами, що спеціалізуються на підготовці тестувальників. Інтеграція продукту в навчальні матеріали дозволить створити нове покоління користувачів, які почнуть знайомство з автоматизацією саме через пропоновану платформу. Такий підхід сприяє формуванню лояльної аудиторії, яка вже на етапі навчання привчається працювати з продуктом, що в майбутньому забезпечує ширше його використання в професійній діяльності. Крім того, співпраця з освітніми організаціями підвищує видимість стартапу та підсилює його репутацію в індустрії.

Оскільки продукт є SaaS-рішенням, важливе значення має політика ціноутворення. Вона повинна бути гнучкою та враховувати широкий спектр можливих користувачів: від студентів і фрилансерів до великих корпорацій. На початковому етапі доцільно застосовувати модель freemium, яка дає змогу користувачу протестувати базовий функціонал без фінансових зобов’язань. Такий підхід дозволяє сформувати перше коло прихильників і перетворити частину з них

у платних користувачів після того, як вони переконаються у практичності інструмента. Для бізнес-клієнтів може бути запропонована система тарифів зі збільшеним функціоналом, підтримкою командної роботи, розширеною кількістю тестів, а також можливістю приватної інтеграції AI-модуля у внутрішню інфраструктуру компанії.

Важливим напрямом маркетингової програми стане побудова бренду, який асоціюється з простотою, швидкістю й технологічністю. Комунікаційна стратегія повинна підкреслювати революційність підходу до автоматизації: замість тривалого й складного процесу написання коду користувач отримує можливість створювати автоматизовані тести за кілька секунд. Саме це є головним емоційним тригером, який формує інтерес і стимулює аудиторію до взаємодії з продуктом. Усі рекламні матеріали мають орієнтуватися на демонстрацію реальної економії часу та ресурсів, що створює сильну конкурентну перевагу.

Завершальним аспектом маркетингової програми є підтримання довгострокових відносин з користувачами. Регулярні оновлення функціоналу, покращення AI-модуля, розширення інструментів інтеграції, участь у професійних подіях і швидка реакція на запити та зворотний зв’язок є основою успішного зростання стартапу. Усе це дозволяє формувати культуру постійного вдосконалення продукту та підвищувати рівень довіри з боку користувачів.

Отже, маркетингова програма стартап-проєкту має комплексний характер і спрямована на поступове, але впевнене формування власної ринкової ніші. Завдяки поєднанню демонстраційної цінності продукту, цифрової присутності, гнучкої політики ціноутворення та ефективної комунікації зі спільнотами стартап отримує реальні шанси на стійке зростання та довготривалий комерційний успіх.

Висновок до розділу

У результаті проведеного аналізу можна зробити висновок, що запропонований стартап-проєкт має високий потенціал ринкової комерціалізації та реальні перспективи практичного впровадження. Технологічна ідея створення AI-платформи для автоматичної генерації Cypress-тестів є актуальною й відповідає

сучасним тенденціям розвитку інформаційних технологій, де ключовою цінністю стає швидкість, ефективність і мінімізація витрат на рутинні процеси. Запропонований підхід дозволяє значно спростити процес автоматизації тестування, зробити його доступним ширшому колу користувачів і знизити залежність від вузькоспеціалізованих технічних компетенцій.

Проведений огляд ідеї проєкту доводить, що використання природної мови як механізму опису тестових сценаріїв створює унікальну можливість для залучення нової аудиторії, яка раніше не могла ефективно працювати з інструментами автоматизації. Це відкриває потенціал для інтеграції продукту в освітні процеси, внутрішні корпоративні тренінги та інфраструктурні рішення компаній, що прагнуть підвищити рівень автоматизації без суттєвих фінансових чи організаційних витрат. Можливість використання Cypress як основного фреймворку забезпечує високу стабільність, масштабованість і підтримку з боку технічної спільноти, що зміцнює позицію стартапу на ринку.

Технологічний аналіз продемонстрував, що всі інструменти, необхідні для реалізації проєкту, вже доступні, добре документовані й підтримуються міжнародними спільнотами. Це робить реалізацію проєкту не лише можливою, а й оптимальною з точки зору витрат і часу. Комбінація Node.js, Cypress та сучасних AI-моделей створює гнучку й ефективну платформу, здатну адаптуватися до різних типів вебзастосунків і вимог користувачів. Продукт може бути масштабований без суттєвих технологічних перешкод, що робить його придатним як для індивідуальних користувачів, так і для великих команд.

Ринковий аналіз показав, що попит на автоматизацію тестування зростає, а разом з ним зростає й попит на інструменти, які допомагають зробити цей процес швидшим та простішим. Стартап орієнтується на сегмент, який сьогодні не повністю покритий існуючими рішеннями, оскільки більшість конкурентних продуктів є орієнтованими на великі компанії, мають значну ціну або пропонують закриту архітектуру, що обмежує можливості користувача. Запропонована платформа натомість пропонує доступність, зрозумілість і можливість гнучкого застосування в різних галузях, що робить її привабливою для малого й середнього бізнесу.

Розроблена ринкова стратегія демонструє, що стартап має чітке бачення своєї аудиторії та шляхів виходу на ринок. Зосередження на реальній цінності продукту, активна цифрова присутність, інтеграція до освітніх програм, співпраця з QA-спільнотами та прозора демонстрація ефективності створюють міцний фундамент для подальшого зростання. Модель гнучкого ціноутворення дозволяє залучити як індивідуальних користувачів, так і бізнес-клієнтів, що сприяє розвитку продукту та його стабільності на ринку.

Отже, стартап-проєкт має всі необхідні передумови для успішного запуску й подальшого розвитку. Він відповідає актуальним потребам індустрії, має технологічну здійсненність, значний ринковий потенціал і здатність адаптуватися до змін ринкового середовища. Поєднання технологічної інновації, високої корисності для користувача та конкурентоспроможності робить цей проєкт перспективним та придатним для впровадження як комерційного продукту.

ВИСНОВКИ

У магістерській дипломній роботі було досліджено сучасні підходи до автоматизованого тестування вебзастосунків та розроблено фреймворк, що поєднує можливості Cypress із технологіями штучного інтелекту. У теоретичній частині проаналізовано модель Page Object, принципи побудови стабільних тестів і ключові проблеми, які виникають під час масштабування автоматизації. Це створило основу для практичної реалізації програмного рішення.

У межах практичної частини було створено повноцінний Cypress-фреймворк із централізованою структурою сторінок, тестами для основних бізнес-функцій, системою логування, відеозаписом і формуванням Mochawesome-звітів. Реалізовані тести охопили сценарії авторизації, перегляду товарів, сортування, додавання в кошик і оформлення покупки. Отримана інфраструктура продемонструвала стабільність і відтворюваність результатів у різних умовах виконання.

Ключовим результатом стало впровадження AI-модуля, здатного автоматично генерувати Cypress-тести на основі текстового опису користувача. Порівняння ручного й автоматичного створення сценаріїв показало суттєву перевагу AI-підходу: час генерації тесту скоротився з 12–20 хвилин до кількох секунд. Це свідчить про 170–300-кратне зростання продуктивності та підтверджує ефективність інтеграції інтелектуальних технологій у процес тестування.

У роботі також сформовано стартап-проєкт, спрямований на ринкову комерціалізацію AI-платформи. Аналіз ринку, визначення потенційних користувачів, стратегія охоплення та маркетингова програма показали реальні можливості виходу продукту на ринок та його перспективність у сучасній індустрії QA. Доступність, швидкість та простота використання забезпечують конкурентні переваги у порівнянні з існуючими рішеннями.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Vasiljevic S. Mastering Cypress: End-to-End Testing for Web Applications. United States : 2023.
2. Mohan G. Full Stack Testing: A Practical Guide for Delivering High Quality Software. United States : 2022.
3. Jose B. Test Automation. United Kingdom : 2021.
4. Gelfenbuim L. Web Testing with Cypress: Run End-to-End tests, Integration tests, Unit tests across web apps, browsers and cross-platforms. India : 2022.
5. Aniche M. Effective Software Testing. 2022.
6. Da Costa L. Testing JavaScript Applications. 2021.
7. Mwaura W. End-to-End Web Testing with Cypress. 2021.
8. Leloudas P. Introduction to Software Testing: A Practical Guide to Testing, Design, Automation, and Execution. 2023.
9. Homes B. Fundamentals of Software Testing. 2024.
10. Leloudas P. Software Testing Strategies. 2024.
11. Winter A., Meaney R. Team Guide to Software Testability. United Kingdom : 2021.
12. Brown S., Bierig R., Galván E., Timoney J. Essentials of Software Testing. 2021.
13. Why Cypress? [Електронний ресурс]. – Доступний з: https://docs.cypress.io/app/get-started/why-cypress (дата звернення: 04.09.2025).
14. Types of software testing. [Електронний ресурс]. – Доступний з: https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing (дата звернення: 22.10.2025).
15. Cypress Testing Guide. [Електронний ресурс]. – Доступний з: https://testgrid.io/blog/cypress-testing/ (дата звернення: 11.11.2025).
16. Introduction to Cypress Testing Framework. [Електронний ресурс]. – Доступний з: https://www.geeksforgeeks.org/introduction-to-cypress-testing-framework/ (дата звернення: 03.10.2025).
17. The QA Role in Modern SDLC. [Електронний ресурс]. – Доступний з: https://testomat.io/blog/the-qa-role-in-modern-software-development-lifecycle/ (дата звернення: 27.09.2025).

18. Swag Labs Demo App. [Електронний ресурс]. – Доступний з: https://www.saucedemo.com (дата звернення: 15.11.2025).
19. Mochawesome Documentation. [Електронний ресурс]. – Доступний з: https://www.npmjs.com/package/mochawesome (дата звернення: 09.09.2025).
20. Cypress community posts. [Електронний ресурс]. – Доступний з: https://dev.to/t/cypress (дата звернення: 01.11.2025).
21. Cypress Tutorial by Guru99. [Електронний ресурс]. – Доступний з: https://www.guru99.com/cypress-tutorial.html (дата звернення: 19.09.2025).
22. Selenium Documentation. [Електронний ресурс]. – Доступний з: https://www.selenium.dev/ (дата звернення: 05.11.2025).
23. Playwright Documentation. [Електронний ресурс]. – Доступний з: https://playwright.dev/ (дата звернення: 08.10.2025).
24. TestCafe Documentation. [Електронний ресурс]. – Доступний з: https://testcafe.io/ (дата звернення: 14.09.2025).
25. Data Structures for QA Automation Engineers. [Електронний ресурс]. – Доступний з: https://medium.com/@dneprokos/data-structures-for-qa-automation-engineers-993135928456 (дата звернення: 29.10.2025).
26. QA Automation Guide. [Електронний ресурс]. – Доступний з: https://testgrid.io/blog/qa-automation-guide/ (дата звернення: 17.10.2025).
27. Better JavaScript E2E Testing with Cypress. [Електронний ресурс]. – Доступний з: https://medium.com/@valentinog/better-javascript-end-to-end-testing-with-cypress-7d19fc889951 (дата звернення: 25.09.2025).
28. Cypress JavaScript Learning Hub. *[Електронний* ресурс]. – Доступний з: https://www.lambdatest.com/learning-hub/cypress-javascript (дата звернення: 30.10.2025).
29. Test Cases for Automated Tests. [Електронний ресурс]. – Доступний з: https://www.browserstack.com/guide/test-cases-for-automated-tests (дата звернення: 06.11.2025)
30. Benefits of Automated Testing. [Електронний ресурс]. – Доступний з: https://www.telerik.com/blogs/5-benefits-automated-testing-why-important (дата звернення: 12.09.2025)

ДОДАТОК А

server.js

```
require("dotenv").config();
const express = require("express");
const cors = require("cors");
const path = require("path");
const { spawn, exec } = require("child_process");
const cypress = require("cypress");
const { generateTest } = require("./ai-generator");

const app = express();
const PORT = process.env.PORT || 3000;

app.use(cors());
app.use(express.json({ limit: "1mb" }));

const uiDir = path.join(__dirname, "ui");
app.get("/ui", (_req, res) => res.sendFile(path.join(uiDir, "index.html")));
app.use("/ui", express.static(uiDir));
app.get("/", (_req, res) => res.redirect("/ui"));

app.post("/api/generate", async (req, res) => {
  try {
    const { testName, baseUrl, scenario } = req.body || {};
    const result = await generateTest({ testName, baseUrl, scenario });
    res.json({
      ok: true,
      ...result,
      relativePath: `cypress/e2e/ai/${result.fileName}`,
    });
  } catch (err) {
    res.status(400).json({ ok: false, error: err.message || "Generation failed" });
  }
});

app.post("/api/run", async (req, res) => {
  try {
    const { specRelativePath } = req.body || {};
    if (!specRelativePath) throw new Error("specRelativePath is required");

    const spec = path.resolve(process.cwd(), specRelativePath);
    console.log(" Running Cypress test:", spec);

    const results = await cypress.run({
      spec,
      browser: "electron",  // можна 'chrome' якщо треба
      headed: false,
      quiet: true,
      video: false,
    });

    const summary = {
      status:
        results.status ||
        (results.totalFailed > 0
          ? "failed"
          : results.totalPassed === results.totalTests
          ? "passed"
          : "unknown"),
```

```
      totalTests: results.totalTests,
      totalPassed: results.totalPassed,
      totalFailed: results.totalFailed,
      totalDuration: results.totalDuration,
      runs: (results.runs || []).map((r) => ({
        specName: r.spec?.name,
        stats: r.stats,
        tests: (r.tests || []).map((t) => ({
          title: (t.title || []).join(" › "),
          state: t.state,
          error: t.displayError || null,
        })),
      })),
    };

    res.json({ ok: true, summary });
  } catch (err) {
    console.error("❌ Cypress run failed:", err);
    res.status(400).json({ ok: false, error: err.message || "Run failed" });
  }
});

app.post("/api/open", (req, res) => {
  try {
    const { specRelativePath, browser = "electron" } = req.body || {};

    const isWindows = process.platform === "win32";
    let command;
    let args;

    if (specRelativePath) {
      const abs = path.resolve(process.cwd(), specRelativePath);

      const specPath = abs.replace(/\\/g, "/");
      if (isWindows) {

        command = `npx cypress open --browser ${browser} --spec "${specPath}"`;
        args = undefined;
      } else {

        command = "npx";
        args = ["cypress", "open", "--browser", browser, "--spec", specPath];
      }
    } else {
      if (isWindows) {
        command = `npx cypress open --browser ${browser}`;
        args = undefined;
      } else {
        command = "npx";
        args = ["cypress", "open", "--browser", browser];
      }
    }

    console.log(" Launching Cypress GUI:", isWindows ? command : `${command} ${args.join(" ")}`);


    const spawnOptions = {
      cwd: process.cwd(),
      detached: true,
      stdio: "ignore", // Ignore stdio so process can run independently
      windowsHide: isWindows, // Hide CMD window on Windows
    };

    if (isWindows) {
```

```
    spawnOptions.shell = true; // Use shell on Windows for npx.cmd
   }

   const child = spawn(command, args, spawnOptions);


   child.on("error", (err) => {
    console.error("Cypress spawn error:", err.message);
   });


   child.unref();


   res.json({ ok: true, launched: true });
  } catch (e) {
   console.error(" Failed to open Cypress:", e);
   res.status(500).json({ ok: false, error: e.message || "Open failed" });
  }
});
console.log("OpenAI key loaded?", !!process.env.OPENAI_API_KEY);
console.log(" CWD:", process.cwd());
app.listen(PORT, () => console.log(`AI Generator running at http://localhost:${PORT}`));
```

ДОДАТОК Б

auth.cy.js

```
import LoginPage from '../pages/LoginPage'

describe('Login Tests', () => {
  beforeEach(() => {
    cy.fixture('users').as('users')
  })

  it('Valid login', { tags: ['@smoke','@critical'] }, function () {
    LoginPage.loginAs(this.users.validUser.username, this.users.validUser.password)
    cy.url().should('include', '/inventory.html')
  })

  it('Locked out user', { tags: ['@regression','@critical'] }, function () {
    LoginPage.loginAs(this.users.lockedUser.username, this.users.lockedUser.password)
    cy.get('[data-test="error"]').should('contain', 'locked out')
  })

  it('Problem user login', { tags: ['@regression','@critical'] }, function () {
    LoginPage.loginAs(this.users.problemUser.username, this.users.problemUser.password)
    cy.url().should('include', '/inventory.html')
  })

  it('Performance glitch user login', { tags: ['@regression','@critical'] }, function () {
    LoginPage.loginAs(this.users.performanceGlitchUser.username, this.users.performanceGlitchUser.password)
    cy.url().should('include', '/inventory.html')
  })
})
describe('Auth negatives', () => {
  beforeEach(() => {
  LoginPage.visit();
  });

  it('SL-004: empty username shows error', { tags: ['@regression','@medium'] }, () => {
    cy.get('[data-test="password"]').type('secret_sauce');
    cy.get('[data-test="login-button"]').click();
    LoginPage.error().should('contain.text', 'Username is required');
  });

  it('SL-005: invalid password shows error', { tags: ['@regression','@medium'] }, () => {
    cy.get('[data-test="username"]').type('standard_user');
    cy.get('[data-test="password"]').type('wrong_password');
    cy.get('[data-test="login-button"]').click();
    LoginPage.error().should('contain.text', 'do not match any user');
  });

  it('SL-006: session persists after reload', { tags: ['@regression','@medium'] }, () => {
    cy.get('[data-test="username"]').type('standard_user');
    cy.get('[data-test="password"]').type('secret_sauce');
    cy.get('[data-test="login-button"]').click();
    cy.url().should('include', '/inventory.html');
    cy.reload();
  cy.url().should('include', '/inventory.html');
  });
});
```

LoginPage.js

```
class LoginPage {
    visit() {
      cy.visit('/')
    }

    fillUsername(name) {
      cy.get('[data-test="username"]').type(name)
    }

    fillPassword(password) {
      cy.get('[data-test="password"]').type(password)
    }

    submit() {
      cy.get('[data-test="login-button"]').click()
    }

    loginAs(user, pass) {
      this.visit()
      this.fillUsername(user)
      this.fillPassword(pass)
      this.submit()
    }
    error() {
    return cy.get('[data-test="error"]')
  }
  }

export default new LoginPage()
```

generated-smoke-on-swag-labs-20251119-200431.cy.js

```
describe("Smoke on Swag Labs", () => {
  it("should complete a purchase successfully", () => {
    cy.visit("https://www.saucedemo.com");
    cy.get("#user-name").type("standard_user");
    cy.get("#password").type("secret_sauce");
    cy.get("#login-button").click();
    cy.contains(".title", "Products").should("be.visible");
    cy.contains(".inventory_item", "Sauce Labs Bike Light")
      .contains("button", "Add to cart")
      .click();
    cy.get(".shopping_cart_link").click();
    cy.get(".cart_item").should("have.length", 1);
    cy.contains("button", "Checkout").click();
    cy.get("#first-name").type("Olia");
    cy.get("#last-name").type("Test");
    cy.get("#postal-code").type("79000");
    cy.contains("input", "Continue").click();
    cy.contains(".title", "Checkout: Overview").should("be.visible");
    cy.contains("button", "Finish").click();
    cy.contains("h2.complete-header", "Thank you for your order!").should("be.visible");
  });
});
```

cart.cy.js

```
import InventoryPage from '../pages/InventoryPage'
import CartPage from '../pages/CartPage'

describe('Cart tests', () => {
  Cypress.on('uncaught:exception', (err, runnable) => {
    return false
  })

  beforeEach(() => {
    cy.visit('/')
    cy.get('[data-test="login-button"]').should('be.visible')
    cy.login('standard_user', 'secret_sauce')
  })

  it('Add item to cart', { tags: ['smoke','critical'] },() => {
    InventoryPage.addToCartByName('Sauce Labs Backpack')
    cy.get('.shopping_cart_badge').should('contain', '1')
  })

  it('Remove item from cart', { tags: ['regression','critical'] },() => {
    InventoryPage.addToCartByName('Sauce Labs Backpack')
    InventoryPage.addToCartByName('Sauce Labs Backpack') // This will remove it since it's already in cart
    cy.get('.shopping_cart_badge').should('not.exist')
  })
})
describe('Cart extended', { tags: ['@regression'] }, () => {
  beforeEach(() => {
    cy.visit('/')
    cy.get('[data-test="login-button"]').should('be.visible')
    cy.fixture('users').then(({ validUser }) => cy.login(validUser.username, validUser.password));
  });

  it('SL-020/021: items present with names & prices; remove one keeps others', { tags: ['@regression','@critical'] }, () => {
    const items = ['Sauce Labs Backpack', 'Sauce Labs Bike Light', 'Sauce Labs Bolt T-Shirt'];
    items.forEach(n => InventoryPage.addToCartByName(n));
    InventoryPage.openCart();
    items.forEach(n => CartPage.itemByName(n).should('be.visible'));
    cy.get('.inventory_item_price').should('have.length.at.least', 2);
    CartPage.removeByName(items[1]);
    CartPage.itemByName(items[1]).should('not.exist');
    CartPage.itemByName(items[0]).should('exist');
    CartPage.itemByName(items[2]).should('exist');
  });

  it('SL-022: continue shopping returns to catalog', { tags: ['@smoke','@critical'] }, () => {
    InventoryPage.addToCartByName('Sauce Labs Backpack');
    InventoryPage.openCart();
    CartPage.continueShoppingBtn().click();
    cy.url().should('include', '/inventory.html');
  });
});
```

CartPage.js

```
class CartPage {
    clickCheckout() {
      cy.get('[data-test="checkout"]').click()
    }

    checkoutBtn() {
```

```
    return cy.get('[data-test="checkout"]')
  }

  itemByName(name) {
    return cy.contains('.cart_item', name)
  }

  removeByName(name) {
    cy.contains('.cart_item', name).find('button').click()
  }

  continueShoppingBtn() {
    return cy.get('[data-test="continue-shopping"]')
  }
}

export default new CartPage()
```