УДК 004.415.2:004.8

ЗАСТОСУВАННЯ LLM ДЛЯ АВТОМАТИЗАЦІЇ СКЛАДАННЯ ТЕСТОВОЇ ДОКУМЕНТАЦІЇ

Кітораги В. О.
kitoragivioleta25@gmail.com
Черкаський державний фаховий бізнес-коледж
Подорошко Д. І.
м. Черкаси, Україна

У сучасній розробці програмних продуктів тестування та контроль якості займають одне з ключових місць. Проте спеціалісти з QA витрачають чималу частину робочого часу на підготовку різноманітної документації − зокрема, чек-листів, тест-кейсів, баг-репортів та тестових сценаріїв. Як зазначають фахівці, формування повноцінного набору тест-кейсів навіть для однієї вимоги може бути трудомістким процесом, що особливо відчутно для тих, хто лише починає працювати у цій сфері [3].

Розвиток великих мовних моделей створив передумови для часткової автоматизації подібних рутинних завдань. Проблема полягає в тому, що наявні на ринку інструменти здебільшого є універсальними та не враховують особливостей QA-процесів і специфіки їхньої документації [2].

У зв'язку з цим дана робота присвячена вивченню підходу до розробки вузькоспеціалізованого веб-застосунку, який на базі великих мовних моделей автоматично генерує три види QA-артефактів з дотриманням прийнятих у галузі стандартів оформлення.

Для обґрунтування необхідності розробки власного рішення було здійснено огляд наявних інструментів на ринку. Серед спеціалізованих платформ варто виділити TestRail та Zephyr Scale, які орієнтовані на зберігання й управління тест-кейсами, втім функції їх автоматичної генерації в них відсутні. Універсальні AI-інструменти − ChatGPT, GitHub Copilot, Claude − певною мірою справляються зі створенням тестової документації, однак якість результату залежить від багатьох змінних [2].

Зокрема, при однаково сформульованому запиті, але з відмінними вхідними даними, модель може видавати результати з різною структурою та наповненням. Це суттєво ускладнює досягнення однорідності документації. Окрім цього, кінцевий результат значно варіюється залежно від того, як саме сформульований запит і який контекст передував поточному діалогу.

Варто також зазначити, що мовні моделі не зберігають інформацію між окремими сесіями. Для отримання стабільних і узгоджених відповідей необхідно працювати в межах одного діалогу, оскільки при відкритті нового всі попередні уточнення втрачаються. Це знижує зручність використання при тривалій роботі, а безкоштовні версії AI-інструментів додатково обмежують кількість доступних запитів.

Нарешті, існує ризик генерації недостовірної інформації або хибної інтерпретації запиту користувача. У подібних ситуаціях виникає необхідність повторного формулювання та деталізації запиту.

Розроблений застосунок являє собою односторінковий веб-додаток, побудований на основі Next.js 16 у поєднанні з React 19 та TypeScript. Вибір такої архітектури обумовлений можливістю об'єднати клієнтську та серверну логіку в єдиній кодовій базі, що значно спрощує процес розгортання й усуває потребу в окремому серверному компоненті. Генерація артефактів здійснюється за допомогою моделі llama-3.3-70b-versatile, розгорнутої через платформу Groq Cloud. Завдяки апаратним LPU-прискорювачам швидкість обробки становить від 200 до 500 токенів на секунду [2].

Функціонал застосунку охоплює три окремі режими роботи. Перший орієнтований на формування тест-кейсів: користувач надає назву функціональної вимоги, її опис та критерії приймання, після чого система генерує від 10 до 18 структурованих записів, кожен з яких містить унікальний ідентифікатор, передумови, покрокові дії, очікувані результати, пріоритет у діапазоні P0–P3 та відповідні теги. Другий режим призначений для оформлення баг-репортів: довільний опис виявленого дефекту перетворюється на стандартизований документ із визначеним рівнем серйозності (S1–S4) та

рекомендаціями щодо додаткових матеріалів. Третій режим дозволяє генерувати тест-ідеї для API: на вхід подається HTTP-ендпоінт, а на виході формується перелік сценаріїв, що охоплює як позитивні, так і негативні випадки, а також готові curl-команди для їх виконання [2].

Для забезпечення стабільного отримання структурованих даних від мовної моделі реалізовано багаторівневий механізм обробки відповідей, що складається з чотирьох послідовних етапів: безпосереднє перетворення у JSON, вилучення вмісту з Markdown-блоків коду, пошук JSON-структури за допомогою регулярного виразу та посимвольний розбір рядка. У випадку, коли жоден із зазначених методів не дає результату, застосунок автоматично формує повторний запит із відповідним коригуючим повідомленням [2].

Якість генерованих документів значною мірою визначається тим, як побудовано системний промпт. У ньому зафіксовано перелік небажаних формулювань − зокрема, розмитих описів на кшталт «має з'явитися помилка» без вказівки конкретного тексту та елемента інтерфейсу − а також вимоги до кожного поля та еталонні приклади у форматі few-shot. Додавання двох зразкових тест-кейсів безпосередньо до промпту помітно підвищило точність очікуваних результатів: середня оцінка якості на однакових вхідних даних зросла з 6,5 до 8,5 за шкалою від 1 до 10.

З метою запобігання надмірному навантаженню на сервіс впроваджено механізм обмеження запитів − не більше десяти звернень на хвилину з однієї IP-адреси, реалізований через структуру Map у пам'яті процесу. Персональні дані користувача зберігаються виключно на стороні клієнта у localStorage браузера: система підтримує до 50 записів в історії та автоматично очищає сховище при досягненні порогу у 4 МБ, а також зберігає контекст поточного проєкту та чернетки заповнених форм.

Практичну перевірку застосунку було здійснено шляхом порівняння витрат часу на ручне створення QA-документації та з використанням розробленого інструменту. Результати показали суттєве скорочення часових витрат у всіх трьох режимах. Формування набору з 14 тест-кейсів для однієї

функції вручну потребує від 45 до 60 хвилин, тоді як із застосуванням асистента цей процес займає 5–8 хвилин, що відповідає економії близько 87%. Час на оформлення баг-репорту зменшився з 10–15 до 2–3 хвилин (приблизно 80%), а підготовка ідей для тестування API-ендпоінту − з 20–30 до 3–5 хвилин (близько 85%). Застосунок є загальнодоступним та розміщений за адресою https://qa-help-eight.vercel.app/assistant [2].

Окремо варто відзначити освітній потенціал інструменту. Для тих, хто лише знайомиться з QA-практиками й не має досвіду складання тест-кейсів, згенеровані артефакти можуть виконувати роль наочних зразків, що демонструють прийняті у галузі стандарти оформлення. Для більш досвідчених користувачів ті самі матеріали стають зручним орієнтиром або шаблоном, що дозволяє пришвидшити та впорядкувати процес створення документації.

Проведене дослідження підтверджує, що вузькоспеціалізований QA-асистент на базі LLM демонструє вищу ефективність порівняно з універсальними AI-інструментами. Ключовими перевагами є передбачуваність структури вихідних даних, відповідність галузевим стандартам оформлення та менша залежність якості результату від точності формулювання запиту з боку користувача.

Розроблений інструмент дозволяє скоротити час на підготовку тестової документації на 80–87% і водночас забезпечує її формальну стандартизацію. Серед перспективних напрямів подальшого вдосконалення − підключення до платформ управління тестуванням, зокрема TestRail та Jira, впровадження RAG-компонента для роботи з проєктною документацією та специфікаціями, а також розширення функціоналу у бік автоматичної генерації тестових скриптів на основі Playwright або Cypress.

Список використаних джерел:

1. Fan A., Gokkaya B., Harman M. та ін. Large Language Models for Software Engineering: Survey and Open Problems. arXiv:2310.03533. 2023. URL: https://arxiv.org/abs/2310.03533 (дата звернення: 17.03.2026).

2. Кітораги В. О. Програмний асистент для підтримки роботи тестувальників : кваліфікаційна робота. Черкаси : ЧДБК, 2026.
3. Myers G. J., Sandler C., Badgett T. The Art of Software Testing. 3rd ed. Hoboken : John Wiley & Sons, 2011. 240 p.
4. Wang J., Huang Y., Chen C. та ін. Software Testing with Large Language Model: Survey, Landscape, and Vision. arXiv:2307.07221. 2023. URL: https://arxiv.org/abs/2307.07221 (дата звернення: 17.03.2026).
5. Hnatushenko V. V., Pavlenko I. V. Використання генеративного штучного інтелекту в тестуванні програмного забезпечення. Системні технології. 2024. Вип. 2(151). С. 10–20.

УДК 004.4

ОПТИМІЗАЦІЯ ПРОЦЕСУ АВТОМАТИЗОВАНОГО WEB-ПАРСИНГУ ЗА ДОПОМОГОЮ БАГАТОПОТОКОВОСТІ

Прудиус В.М.
agyshii@gmail.com
Черкаський державний технологічний університет
Метелап В.В.
м. Черкаси, Україна

Сучасний етап розвитку інформаційних технологій характеризується експоненційним зростанням обсягів даних у мережі Інтернет. Для аналізу ринку, машинного навчання, агрегації новин та інших науково-дослідних завдань критично важливим є інструментарій автоматизованого збору інформації, відомий як web-парсинг (web scraping). Традиційний однопотоковий підхід до реалізації подібних алгоритмів має суттєвий недолік – він характеризується значними часовими витратами та низьким коефіцієнтом корисної дії апаратного забезпечення. Це пов'язано із проблемою синхронного блокування (I/O Bound операції), при якому програма, відправивши мережевий запит, зупиняє своє виконання і більшу частину часу просто очікує на відповідь від цільового сервера та завантаження HTML-документа, тоді як ресурси центрального процесора (CPU) залишаються абсолютно незадіяними [1].