Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій

Бакалаврська кваліфікаційна робота на тему:

«ЗАСТОСУВАННЯ ТЕХНОЛОГІЇ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ДЛЯ СИСТЕМИ УПРАВЛІННЯ ЛІНГВІСТИЧНИМИ ЗАДАЧАМИ LINGUIST PORTAL»

Виконав: студент IV курсу, групи 1АКІТ-21б
спеціальності 151 – Автоматизація та
комп'ютерно-інтегровані технології
(шифр і назва спеціальності)
Анастасія ЧОРНА
(прізвище та ініціали)

Керівник: д.т.н., професор кафедри АІІТ
Роман КВЄТНИЙ
(прізвище та ініціали)
«16» 06 2025 р.

Рецензент: д.т.н., професор кафедри КСУ
Марія ЮХИМЧУК
(прізвище та ініціали)
«17» 06 2025 р.

Допущено до захисту
завідувач кафедри АІІТ
Олег БІСІКАЛО
«19» 06 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій
Рівень вищої освіти перший (бакалаврський)
Галузь знань 15 – Автоматизація та приладобудування
Спеціальність 151 – Автоматизація та комп'ютерно-інтегровані технології
Освітньо-професійна програма Інтелектуальні комп'ютерні системи управління

ЗАТВЕРДЖУЮ
завідувач кафедри АІІТ
_____ д.т.н., проф. Олег БІСІКАЛО
«19» 06 2025 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Чорній Анастасії Олегівні
(прізвище, ім'я, по батькові)

1. Тема роботи «Застосування технології автоматизованого тестування для системи управління лінгвістичними задачами Linguist Portal»
керівник роботи Квєтний Роман Наумович, д.т.н, професор
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від «20» березня 2025 року №97
2. Термін подання студентом роботи 10.06.2025 р.
3. Вихідні дані до роботи:
   - технічне завдання та функціональні можливості системи Linguist Portal;
   - web інтерфейс та REST API, доступні зареєстрованому користувачу;
   - типові сценарії взаємодії зареєстрованого користувача з системою;
   - інструмент Cypress для автоматизованого тестування UI;
   - інструмент Postman для перевірки API-запитів.
4. Зміст текстової частини (перелік питань, які потрібно розробити) провести аналіз поняття тестування, важливість тестування, наслідки його відсутності, класифікації методів та видів тестування, життєвий цикл тестування; огляд автоматизованого тестування, дослідження існуючих інструментів для автоматизованого тестування та їх класифікації; провести автоматизоване тестування веб інтерфейсу та API для системи управління лінгвістичними задачами Linguist Portal.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень) Автоматизоване тестування web інтерфейсу в середовищі Cypress; автоматизоване тестування API в середовищі Postman; варіанти використання Linguist Portal; діаграма класів Linguist Portal; діаграма активності автоматизованого тестування Linguist Portal.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завд прий
1-4	Квєтний Р.Н., д.т.н., проф.		

7. Дата видачі завдання 20.03.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Термін виконання		При
		початок	закінчення	
1	Вибір, узгодження та затвердження теми БКР	03.10.2024	09.10.2024	
2	Аналіз предметної області та опрацювання літературних джерел.	12.03.2025	29.03.2025	
3	Постановка задач для вирішення в БКР	01.04.2025	26.04.2025	
4	Вирішення поставлених задач	29.04.2025	10.05.2025	
5	Підтвердження достовірності отриманих результатів	13.05.2025	24.05.2025	
7	Оформлення пояснювальної записки та графічної частини	27.05.2025	31.05.2025	
8	Перевірка ПЗ на відповідність вимогам	03.06.2025	07.06.2025	
9	Попередній захист, доопрацювання, та рецензування БКР	10.06.2025	15.06.2025	
10	Захист БКР	16.06.2025	23.06.2025	

Студент _____ Анастасія ЧОРНА
(підпис) (ім'я та прізвище)

Керівник роботи _____ Роман КВЄТНИЙ
(підпис) (ім'я та прізвище)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 63 сторінок формату А4 в тому числі і додатків, на яких є 3 рисунка, список джерел містить 18 найменувань.

Робота досліджує застосування автоматизованого тестування для системи управління лінгвістичними задачами Linguist Portal. У роботі розкрито поняття тестування та автоматизованого тестування, їхні переваги й особливості. Описано структуру Linguist Portal, що включає web інтерфейс на основі ASP.NET і RESTful API. Проведено тестування web інтерфейсу за допомогою Cypress для перевірки функціональності та стабільності, а також тестування API через Postman для оцінки коректності запитів і відповідей. Робота демонструє ефективність автоматизованого тестування для підвищення якості програмного забезпечення.

Ключові слова: вебдодаток, web інтерфейс, автоматизоване тестування, API.

ABSTRACT

The bachelor`s qualification thesis consists of 63 A4 pages, including appendices, and contains 3 figures. The list of reference includes 18 sources.

The thesis explores the application of automated testing for the Linguist Portal linguistic task management system. It presents the concepts of testing and automated testing, highlighting their benefits and specific features. The structure of the Linguist Portal is described, including a web interface based on ASP.NET and a RESTful API. Web interface testing was conducted using Cypress to verify functionality and stability, while API testing was performed with Postman to assess the correctness of requests and responses. The thesis demonstrates the effectiveness of automated testing in improving software quality.

Keywords: web application, web interface, automated testing, API.

ЗМІСТ

ВСТУП

У сучасному світі інформаційних технологій розробка програмного забезпечення є складним і багатогранним процесом, що вимагає забезпечення високої якості продуктів для задоволення потреб користувачів. Одним із ключових етапів розробки є тестування, яке дозволяє виявляти та усувати помилки, забезпечуючи надійність і стабільність програмних систем. З появою складних систем, таких як платформи для управління лінгвістичними задачами, зростає потреба у впровадженні автоматизованого тестування, яке значно підвищує ефективність і швидкість перевірки програмного забезпечення порівняно з ручними методами. Автоматизоване тестування дозволяє не лише економити час і ресурси, але й забезпечує систематичний підхід до перевірки складних компонентів, таких як web інтерфейси та API, що є особливо актуальним для сучасних вебдодатків.

Актуальність роботи зумовлена зростаючою складністю програмних систем, зокрема систем управління лінгвістичними задачами, таких як Linguist Portal. Ця платформа, призначена для автоматизації процесів перекладу, локалізації та управління текстовими даними, вимагає ретельного тестування для забезпечення її надійності та зручності використання. Впровадження автоматизованого тестування дозволяє виявляти помилки на ранніх етапах розробки, зменшувати витрати на виправлення дефектів і підвищувати якість продукту. Використання сучасних інструментів, таких як Cypress для тестування web інтерфейсу та Postman для тестування API, відповідає актуальним тенденціям у сфері розробки програмного забезпечення, що робить дослідження цієї теми важливим як із теоретичної, так і з практичної точки зору.

Метою роботи є підвищення якості та надійності системи управління лінгвістичними задачами Linguist Portal шляхом поглибленого аналізу методів та інструментів автоматизованого тестування та їх практичного застосування.

Об’єкт дослідження — процес автоматизованого тестування програмного забезпечення, зокрема системи Linguist Portal, яка включає web інтерфейс на основі ASP.NET і RESTful API.

Предмет дослідження — методи та інструменти автоматизованого тестування, зокрема Cypress для тестування web інтерфейсу та Postman для тестування API, застосовані до системи Linguist Portal.

Завдання роботи, які необхідно розв’язати для досягнення поставленої мети:

1. Проаналізувати теоретичні основи тестування та автоматизованого тестування програмного забезпечення.
2. Описати структуру системи Linguist Portal, включаючи її web інтерфейс і API.
3. Обґрунтувати вибір інструментів автоматизованого тестування (Cypress і Postman) для перевірки системи.
4. Розробити та провести тести web інтерфейсу Linguist Portal за допомогою Cypress, перевіривши ключові сценарії користувача.
5. Виконати тестування API системи за допомогою Postman, оцінивши коректність обробки запитів і відповідей.
6. Оцінити ефективність застосованих методів автоматизованого тестування та їхній вплив на якість системи.

Методи дослідження включають аналіз літератури з питань тестування програмного забезпечення, порівняльний аналіз інструментів автоматизованого тестування, експериментальне тестування web інтерфейсу та API системи Linguist Portal, а також оцінку результатів тестування для визначення їхньої ефективності.

Практична цінність роботи полягає у розробці та впровадженні набору автоматизованих тестів для системи Linguist Portal, що дозволяє підвищити її якість, зменшити кількість помилок і оптимізувати процеси розробки. Отримані результати можуть бути використані для вдосконалення процесів тестування в подібних системах управління лінгвістичними задачами, а також як основа для подальших досліджень у сфері автоматизованого тестування.

1 ОГЛЯД МЕТОДІВ ТЕСТУВАННЯ ВЕБДОДАТКІВ

1.1 Що таке тестування вебдодатків

У добу бурхливого розвитку цифрових технологій, коли вебдодатки відіграють вирішальну роль у різних галузях — від бізнесу до освіти, — забезпечення їхньої високої якості набуває надзвичайної важливості. Одним із ключових засобів підтримки якості є тестування вебдодатків, яке дозволяє виявляти можливі вади ще до того, як продукт стане доступним для кінцевих користувачів.

Тестування вебдодатків охоплює оцінку функціональності, стабільності, продуктивності, безпеки та загальної відповідності системи встановленим технічним вимогам [1]. Його головна мета — своєчасне виявлення недоліків, зменшення ймовірності збоїв і гарантування високого рівня якості продукту. У складних цифрових екосистемах, де вебдодатки включають клієнтську частину (front-end), серверну логіку (back-end), бази даних і численні зовнішні сервіси, тестування стає невід'ємною умовою для стабільної роботи всіх елементів системи.

Архітектура вебдодатків вирізняється значною складністю через багатошаровість, що охоплює клієнтську частину, серверну логіку, бази даних та інтеграцію з зовнішніми сервісами. Така складна структура робить вебдодатки вразливими до помилок, навіть незначних, які можуть спричинити серйозні проблеми. Наприклад, дрібний недолік у коді може призвести до втрати даних користувачів, що загрожує функціональній цілісності системи та її репутації. Крім того, помилки можуть викликати зниження продуктивності, як-от повільна обробка запитів, що ускладнює взаємодію користувачів, або порушення роботи інтерфейсу, коли елементи відображаються некоректно чи певні функції стають недоступними.

Тестування вебдодатків є ключовим інструментом для мінімізації подібних ризиків, дозволяючи виявляти та виправляти помилки на ранніх етапах розробки або перед розгортанням системи. Систематичні перевірки, такі як наскрізні тести

для оцінки інтерфейсу чи перевірка серверних запитів, забезпечують стабільність і коректність роботи всіх компонентів. Це не лише запобігає технічним збоям, але й сприяє покращенню користувацького досвіду, що є критично важливим для вебдодатків, де зручність і інтуїтивність інтерфейсу безпосередньо впливають на задоволеність користувачів. Наприклад, тестування сценаріїв авторизації чи обробки форм гарантує, що користувачі можуть безперешкодно виконувати свої завдання. Тестування також підвищує надійність вебдодатку, забезпечуючи його здатність стабільно працювати за умов високого навантаження чи різноманітних сценаріїв використання.

На ширшому рівні тестування виконує не лише технічну функцію, спрямовану на усунення недоліків, але й стратегічну роль, забезпечуючи відповідність вебдодатку високим стандартам якості, безпеки та зручності. Якість досягається завдяки ретельній перевірці всіх аспектів системи, від серверної логіки до відображення елементів у браузері. Безпека гарантується тестами, що виявляють потенційні вразливості, наприклад, у процесі авторизації чи обробки даних. Зручність досягається через оптимізацію інтерфейсу, перевіреного шляхом імітації реальних сценаріїв взаємодії. Таким чином, тестування стає невід'ємною частиною процесу розробки, сприяючи створенню стабільних, ефективних і конкурентоспроможних вебдодатків, які відповідають сучасним вимогам до якості програмного забезпечення [2].

1.2 Важливість тестування для якості продукту

Якість програмного забезпечення є складною багатовимірною характеристикою, яка охоплює не лише технічну досконаліть продукту, а й його здатність відповідати очікуванням і потребам кінцевого користувача. Визначення якості не обмежується відсутністю помилок у коді чи стабільністю виконання. Сучасне розуміння цього поняття включає ширший спектр параметрів, таких як зручність використання інтерфейсу (usability), швидкодія системи (performance), адаптив-

ність до різних пристроїв та платформ (responsiveness), захищеність даних та користувачів (security), масштабованість, підтримуваність і надійність. Усі ці аспекти тісно пов’язані з користувацьким досвідом і загальним враженням від використання програмного продукту.

Таким чином, якість програмного забезпечення — це не лише результат правильно написаного коду, а сукупність властивостей, які формують цінність продукту для його аудиторії. Якщо хоча б один із важливих нефункціональних параметрів буде реалізований неякісно, це може суттєво знизити рівень задоволеності користувача, навіть якщо основна функціональність працює без збоїв.

У цьому контексті тестування виконує вирішальну роль у процесі забезпечення якості. Воно дозволяє виявити та усунути потенційні проблеми ще до релізу продукту, тобто до того моменту, коли додаток стане доступним для широкого загалу користувачів. Це не лише підвищує якість, а й дозволяє оптимізувати витрати на подальшу підтримку. Як відомо з практики розробки програмного забезпечення, вартість виправлення помилок зростає експоненційно залежно від етапу, на якому вона була виявлена. Наприклад, якщо помилка буде знайдена на стадії вимог або проєктування, її усунення потребуватиме мінімальних ресурсів. У той час як аналогічна помилка, виявлена вже після запуску продукту в експлуатацію, може вимагати значного обсягу робіт, пов’язаних із рефакторингом, оновленням інтерфейсу або навіть зміною архітектури.

Крім економічної доцільності, тестування сприяє підвищенню конкурентоспроможності продукту. У сучасному цифровому середовищі, де користувачі мають широкий вибір між аналогічними програмними рішеннями, якість є тим критерієм, що часто визначає успіх або провал проєкту. Вчасне та якісне тестування забезпечує стабільну роботу продукту, позитивний користувацький досвід і, відповідно, формує довіру до бренду чи компанії-розробника [2].

Таким чином, можна зробити висновок, що тестування — це не просто етап розробки, а важлива інвестиція в якість, економічну ефективність та ринкову привабливість програмного забезпечення.

1.3 Тестування як невід’ємна частина розробки

У межах сучасних методологій розробки програмного забезпечення, таких як Agile, Scrum і DevOps, відбувається перегляд ролі тестування в життєвому циклі програмного продукту. Якщо раніше тестування вважалося переважно завершальним етапом після створення основного функціоналу, то нині воно інтегрується як невід’ємна частина всіх етапів розробки — від аналізу вимог до супроводу продукту після випуску [11].

Застосування гнучких методологій розробки програмного забезпечення, зокрема Agile та Scrum, передбачає ітеративний підхід до створення продукту, в межах якого розробка здійснюється короткими циклами, або спринтами. Кожен із таких спринтів завершується створенням інкременту — завершеного фрагмента функціональності, який повинен бути повністю готовим до використання, тобто таким, що пройшов усі необхідні етапи перевірки. У цьому контексті тестування перестає бути окремою завершальною фазою проєкту і перетворюється на безперервний процес, інтегрований у всі стадії життєвого циклу розробки.

Однією з ключових особливостей гнучких підходів є раннє залучення тестувальників до роботи над продуктом. Вони беруть участь ще на етапі аналізу вимог і планування ітерацій, що дозволяє не лише краще розуміти бізнес-цілі, але й формулювати тестові сценарії ще до написання коду. Такий підхід має суттєві переваги: вже на ранньому етапі виявляються логічні неузгодженості, неповні або суперечливі вимоги, а також потенційні технічні ризики, які можуть вплинути на реалізацію майбутнього функціоналу. Завдяки цьому проблеми усуваються до того, як вони перетворяться на дефекти в програмному коді.

Інтеграція тестування у кожен спринт забезпечує постійний зворотний зв’язок і дозволяє підтримувати високий рівень якості протягом усього проєкту. Використання автоматизованих тестів у такому підході є особливо ефективним, адже воно дає змогу швидко перевіряти стабільність нових змін, не витрачаючи час на повторне ручне тестування. Внаслідок цього значно скорочуються ви-

трати на виправлення помилок, оскільки вони виявляються та усуваються на ранніх етапах, а також підвищується гнучкість і швидкість реагування на зміну вимог.

Підхід DevOps, який поєднує процеси розробки (Development) та операційної підтримки (Operations), також наголошує на інтеграції тестування в повсякденний робочий цикл. Центральну роль тут відіграє впровадження конвеєрів безперервної інтеграції та розгортання (CI/CD — Continuous Integration / Continuous Deployment). Ці практики включають автоматизовану перевірку коду на кожному етапі — від внесення змін до репозиторію до випуску оновлень на продуктивне середовище. Завдяки інтегрованим тестам (unit tests, integration tests, UI tests) забезпечується швидка оцінка якості кожного оновлення з мінімальними затримками та людським втручанням.

Отже, автоматизація тестування в рамках CI/CD сприяє швидкому виявленню та усуненню помилок, знижує ризик появи критичних дефектів у релізних версіях, підвищує стабільність коду та зміцнює впевненість розробників у його працездатності. У підсумку, організація тестування як інтегрованого процесу зменшує витрати на підтримку, прискорює випуск релізів, підвищує надійність програмного забезпечення та покращує користувацький досвід [12].

У сучасному IT-середовищі тестування програмного забезпечення трансформувалося з простого завершального етапу в ключовий стратегічний компонент усього циклу розробки. Воно відіграє центральну роль у забезпеченні не лише технічної бездоганності вебдодатків, але й їхньої відповідності високим стандартам якості, які очікують користувачі. Тестування дозволяє виявляти та усувати потенційні проблеми на ранніх етапах, що сприяє зниженню витрат на виправлення помилок і підвищенню загальної ефективності розробки. Крім того, інтеграція тестування в процес створення продукту забезпечує стабільність, безпеку та зручність використання вебдодатків, що є критично важливим у конкурентному цифровому середовищі. Завдяки такому підходу розробники можуть створювати надійні рішення, які не лише відповідають технічним вимогам, але й забезпечують позитивний користувацький досвід. Таким чином, тестування

стало невід’ємною частиною стратегії розробки, що гарантує успіх проєктів і довгострокову цінність продуктів.

1.4 Наслідки відсутності або недооцінки тестування

Недооцінка чи повне ігнорування процесу тестування та його стратегічного значення у розробці програмного забезпечення може спричинити серйозні, а в окремих випадках катастрофічні наслідки як для самого продукту, так і для компанії-розробника. У сучасному конкурентному цифровому середовищі, де користувачі очікують від додатків не лише функціональності, а й стабільності, безпеки та зручності, збої в роботі вебдодатків можуть мати довготривалі негативні ефекти.

Серед ключових ризиків, пов’язаних із недостатнім тестуванням або його відсутністю, можна виділити:

- Запуск нестабільного чи небезпечного продукту. Без належної перевірки програмне забезпечення може містити критичні вади, які спричинять збої, втрату даних або компрометацію безпеки. Це особливо актуально для вебдодатків із відкритим доступом, що обробляють конфіденційні чи фінансові дані.
- Негативний досвід користувачів і втрата довіри. Погана продуктивність, нестабільна робота, незручний інтерфейс або невідповідність очікуванням можуть призвести до негативних відгуків, зменшення кількості користувачів і втрати клієнтської бази. У світі, де альтернативні сервіси доступні за одним кліком, збереження довіри є критичним, а її втрата — важко відшкодовуваною.
- Фінансові збитки. Збої в роботі додатку можуть спричинити зупинку бізнес-процесів, втрату доходів, витрати на виправлення помилок у реальному часі або компенсації клієнтам. Крім того, розробка виправлень і оновлень після випуску зазвичай потребує значно більше ресурсів, ніж усунення проблем на етапі тестування.
- Пошкодження репутації компанії. Одноразовий інцидент, наприклад, витік даних чи тривале відключення сервісу, може серйозно вплинути на

репутацію компанії, підірвати довіру інвесторів і партнерів, а в крайніх випадках загрожувати її подальшому існуванню.

Історія IT-індустрії вже не раз ілюструвала випадки, коли незначна помилка в вебдодатку призводила до мільйонних збитків. У деяких ситуаціях такі збої спричиняли масове видалення чи витік даних користувачів, в інших — тривалу недоступність сервісу. Відомі інциденти в діяльності компаній, таких як Amazon, Google, Facebook, а також у фінансових установах і державних сервісах, демонструють, наскільки руйнівними можуть бути наслідки відсутності належного тестування [2].

Тому тестування виступає ефективним способом профілактики ризиків, дозволяючи виявляти потенційні проблеми на етапі розробки та усунути їх до потрапляння в продуктивне середовище. Це не лише зменшує ймовірність критичних інцидентів, а й підвищує надійність та довіру до цифрового продукту загалом. У підсумку, тестування є не витратами, а інвестицією в стабільність, безпеку та комерційний успіх продукту на ринку.

1.5 Класифікація методів тестування

Методи тестування програмного забезпечення являють собою сукупність підходів, які використовуються для всебічної оцінки правильності функціонування, надійності роботи та загальної якості розробленої системи. Ці методи дозволяють систематично перевіряти, чи відповідає програмний продукт поставленим вимогам, чи здатен він стабільно виконувати свої функції та чи відповідає очікуванням користувачів. Вибір конкретного методу тестування залежить від низки ключових факторів, зокрема від цілей, які ставляться перед перевіркою, таких як виявлення помилок, забезпечення безпеки чи підтвердження відповідності специфікаціям. Крім того, важливу роль відіграє доступ до вихідного коду системи: деякі методи, як-от тестування «білого ящика», потребують повного доступу до коду, тоді як інші, наприклад тестування «чорного ящика», базуються виключно на зовнішній поведінці системи.

Глибина аналізу, необхідна для тестування, також впливає на вибір методу. Наприклад, поверхневе тестування може бути достатнім на ранніх етапах розробки, тоді як детальний аналіз критичних компонентів необхідний перед релізом продукту. Етап життєвого циклу програмного забезпечення є ще одним визначальним чинником: на етапі розробки можуть застосовуватися модульні тести, тоді як на етапі впровадження чи підтримки доцільними будуть інтеграційні чи регресійні перевірки. Класифікація методів тестування відіграє важливу роль у впорядкуванні цих підходів, дозволяючи тестувальникам чітко визначити, які інструменти та техніки найкраще підходять для конкретного завдання.

Така систематизація сприяє формуванню структурованого підходу до перевірки якості програмного забезпечення та забезпечує можливість вибору оптимальної стратегії тестування, яка відповідає унікальним особливостям кожного проєкту. Наприклад, для складних систем, таких як Linguist Portal, класифікація методів дозволяє обрати комбінацію тестів, що враховує специфіку веб-інтерфейсу та API. Таким чином, правильно підібрана стратегія тестування, заснована на чіткій класифікації методів, сприяє підвищенню ефективності процесу перевірки, зниженню ризиків і забезпеченню високої якості кінцевого продукту, що є критично важливим для успіху проєкту.

Одним із поширених способів класифікації є розподіл методів за рівнем доступу до внутрішньої структури програмного забезпечення. У цьому контексті виділяють методи «чорного ящика», «білого ящика» та «сірого ящика». Метод «чорного ящика» зосереджується на зовнішній поведінці системи: тестувальник не має доступу до коду, а перевірка базується на специфікаціях та очікуваних результатах, імітуючи дії реального користувача. Метод «білого ящика», навпаки, передбачає повний доступ до вихідного коду, що дозволяє аналізувати його логіку, структуру гілок, циклів, умов та винятків, забезпечуючи високий рівень покриття тестами. Метод «сірого ящика» комбінує елементи обох підходів: тестувальник має часткові відомості про внутрішню архітектуру, що полегшує створення тестів для складних сценаріїв.

Інший підхід до класифікації ґрунтується на етапі проведення тестування: методи поділяють на статичні та динамічні. Статичні методи не передбачають запуску програми — вони включають аналіз документації, специфікацій, архітектури чи коду для виявлення потенційних помилок до початку виконання. Наприклад, це може бути рецензування коду, статичний аналіз чи перевірка на відповідність стандартам. Динамічні методи, навпаки, вимагають фактичного запуску програмного забезпечення, що дозволяє виявляти вади, пов’язані з логікою виконання, взаємодією компонентів чи поведінкою в реальних умовах.

Методи тестування також класифікують за рівнем формалізації: формальні методи спираються на математичні моделі й застосовуються, зокрема, в критичних системах (наприклад, авіація чи медицина), де помилки можуть мати серйозні наслідки. Неформальні методи, у свою чергу, базуються на практичних підходах без потреби в математичній точності, що робить їх ефективними для більшості комерційних проєктів [1].

Таким чином, класифікація методів тестування сприяє глибшому розумінню їхнього призначення та потенціалу, а також дозволяє зробити обґрунтований вибір технік залежно від особливостей програмного продукту, цілей тестування та наявних ресурсів. Цей підхід забезпечує гнучкість і ефективність у плануванні процесу перевірки.

1.6 Основні види тестування вебдодатків

Типи тестування вебдодатків охоплюють широкий спектр підходів, які відіграють ключову роль у забезпеченні високої якості, стабільності роботи та зручності використання програмного продукту. Вони дозволяють виявляти недоліки на різних етапах розробки, забезпечуючи надійність і відповідність вебсервісу очікуванням користувачів. Залежно від цілей, особливостей проєкту та його вимог застосовуються різноманітні види тестування, кожен із яких зосереджується на перевірці певних аспектів функціональності, продуктивності чи безпеки вебдодатку. Такий багатогранний підхід допомагає створювати продукти, які не

лише відповідають технічним стандартам, але й забезпечують комфортну взаємодію для кінцевих користувачів. Детальний огляд основних типів тестування розглянемо нижче..

Функціональне тестування перевіряє, чи відповідає поведінка системи встановленим вимогам, зокрема коректність роботи бізнес-логіки, навігації та обробки форм.

Тестування продуктивності оцінює, як система витримує навантаження, визначаючи її швидкодію, стабільність і здатність обробляти одночасно велику кількість запитів.

Безпекове тестування виявляє вразливі місця вебдодатку та перевіряє його стійкість до атак чи несанкціонованого доступу.

Юзабіліті тестування зосереджується на зручності інтерфейсу та загальному досвіді користувача, допомагаючи оцінити інтуїтивність використання додатку.

Кросбраузерне тестування забезпечує правильну роботу вебдодатку в різних браузерах і на різних версіях платформ.

Регресійне тестування проводиться після оновлень коду, щоб переконатися, що нові зміни не вплинули на існуючий функціонал.

Тестування сумісності на мобільних пристроях гарантує коректне відображення та функціональність додатку на екранах різного розміру й операційних системах.

Тестування доступності (accessibility testing) спрямоване на перевірку зручності використання продукту для людей із особливими потребами.

Хоча детальний аналіз кожного з цих типів тестування буде представлено в наступному підрозділі, уже на цьому етапі варто зазначити, що лише комплексне застосування різних видів тестування дає змогу всебічно оцінити як функціональні, так і нефункціональні характеристики вебдодатку. Такий підхід сприяє створенню більш надійного, стабільного та зручного для користувачів кінцевого продукту [1,2].

1.7 Життєвий цикл тестування (STLC)

У межах сучасного програмного забезпечення, де якість та надійність продукту є критично важливими, особливе значення набуває системний підхід до тестування. Саме з цією метою застосовується концепція життєвого циклу тестування — Software Testing Life Cycle (STLC), яка передбачає поетапну організацію процесів, пов'язаних із виявленням, фіксацією та усуненням помилок у програмному забезпеченні.

Життєвий цикл тестування — це не просто набір дій, а добре структурована послідовність етапів, які виконуються для досягнення високого рівня якості продукту. STLC дозволяє формалізувати підхід до перевірки вебдодатків, забезпечуючи повторюваність, контрольованість та передбачуваність результатів тестування. Особливо це важливо у великих проєктах, де взаємодія численних команд вимагає чіткої координації.

Основні етапи STLC охоплюють увесь процес тестування — від аналізу документації до остаточного звітування. Розглянемо кожен із них детальніше:

1. Аналіз вимог

Перший і надзвичайно важливий етап — це аналіз вимог до програмного продукту. У цей період тестувальники ознайомлюються з документацією (включаючи технічне завдання, специфікації, юзер-сторі тощо), виявляють можливі неузгодженості, суперечності чи неповноту. Глибокий аналіз вимог дозволяє не лише визначити, що саме підлягає тестуванню, а й сформулювати чіткі очікування до функціональності та поведінки системи.

2. Планування тестування

Після аналізу вимог відбувається складання тест-плану — документа, що описує стратегію тестування, доступні ресурси, строки, типи тестів, підходи до автоматизації та потенційні ризики. Цей етап визначає загальний напрямок тестування, формує основи для прийняття рішень щодо обсягів робіт і вибору інструментів.

3. Розробка тест-кейсів

Наступним кроком є створення тест-кейсів — сценаріїв, які описують конкретні дії користувача, вхідні дані, очікувані результати та критерії проходження. Добре написані тест-кейси гарантують, що кожен аспект функціональності буде перевірений послідовно та повно. На цьому етапі часто використовуються техніки проектування тестів, такі як еквівалентне розбиття, граничні значення, таблиці прийняття рішень тощо.

4. Підготовка тестового середовища

Щоб забезпечити точність та достовірність результатів тестування, необхідно створити або налаштувати тестове середовище, яке максимально наближене до реального середовища експлуатації. Це може включати налаштування серверів, баз даних, емуляцію зовнішніх сервісів, конфігурацію мережевих умов тощо. Якісне тестове середовище мінімізує ризик появи помилок, спричинених відмінностями між етапами розробки та продакшеном.

5. Виконання тестів

Після підготовки середовища відбувається безпосереднє виконання тестів. Тестувальники проходять розроблені тест-кейси вручну або за допомогою інструментів автоматизації. За результатами тестів фіксуються знайдені дефекти, порівнюються фактичні результати з очікуваними, і формується база для подальшої роботи команди розробки.

6. Звітність і моніторинг дефектів

Зібрані під час тестування дані фіксуються в системах управління дефектами, таких як Jira, TestRail, Bugzilla тощо. Цей етап включає в себе також моніторинг статусу тестів, аналіз причин виникнення помилок, пріоритезацію задач. Регулярна звітність дозволяє стейкхолдерам оцінити поточний стан якості продукту, динаміку тестування та прогрес виправлення дефектів.

7. Завершення тестування

Після завершення усіх запланованих активностей відбувається аналіз виконаної роботи. Команда тестування проводить ретроспективу, оцінює ефективність тестів, виявляє сильні та слабкі сторони процесу. Готуються фінальні звіти,

які включають метрики тестування, статистику знайдених і виправлених дефектів, рекомендації щодо подальшого покращення процесу. Документація, створена на цьому етапі, є важливим джерелом знань для майбутніх проєктів.

Узагальнюючи, можна зазначити, що STLC — це не просто формальність, а дієвий інструмент організації тестування, який дозволяє забезпечити прозорість, контрольованість і ефективність усіх етапів перевірки програмного продукту. Такий структурований підхід особливо цінний у великих проєктах і в середовищі з високими вимогами до якості [1].

2 АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ ВЕБДОДАТКІВ: ПРИНЦИПИ ТА ІНСТРУМЕНТИ

2.1 Поняття автоматизованого тестування

Автоматизоване тестування — це процес перевірки програмного забезпечення, що передбачає використання спеціалізованих інструментів і скриптів для автоматичного виконання тестових сценаріїв, порівняння фактичних результатів із очікуваними та фіксації результатів тестування. Його метою є виявлення помилок у функціонуванні програми, забезпечення її стабільності та відповідності вимогам ще до впровадження у виробниче середовище. Автоматизація охоплює не лише безпосередній запуск тестів, а й створення інфраструктури для їх регулярного виконання, інтеграції з іншими процесами розробки та звітування.

На відміну від ручного тестування, де кожен етап — від підготовки даних і написання тестових сценаріїв до запуску перевірок і аналізу результатів — виконується тестувальником вручну, автоматизоване тестування значно скорочує час і зусилля, необхідні для цих завдань. Програма або фреймворк виконує всі дії самостійно за заздалегідь визначеним алгоритмом, що забезпечує високу швидкість проведення перевірок та послідовність їх виконання. Завдяки автоматизації зменшується ймовірність допущення помилок, пов’язаних із людським фактором, зокрема таких як неточне введення даних, помилкове трактування результатів або пропуск кроків сценарію.

Крім того, автоматизоване тестування дозволяє багаторазово запускати одні й ті самі тест-кейси — як під час розробки нового функціоналу, так і при регресійних перевірках після оновлення коду. Це забезпечує повторюваність результатів та постійний контроль якості програмного продукту без додаткових витрат ресурсів. Завдяки своїй ефективності, масштабованості та здатності інтегруватися з процесами CI/CD, автоматизація стає незамінним елементом сучасної практики забезпечення якості, особливо у великих проєктах або за умови частих оновлень програмного забезпечення [2].

У сучасному процесі розробки програмного забезпечення, особливо в умовах динамічних методологій, таких як Agile чи DevOps, автоматизоване тестування відіграє ключову роль. Воно стало невід'ємною частиною процесів безперервної інтеграції та доставки (CI/CD), дозволяючи оперативно перевіряти зміни в коді ще до їхнього впровадження в продуктивне середовище [11,12]. Завдяки цьому автоматизовані тести допомагають швидко виявляти помилки, спричинені оновленнями, і запобігають їх накопиченню в системі.

Особливо цінною автоматизація є для великих або тривалих проєктів, де кількість тест-кейсів постійно зростає, а регресійне тестування вимагає дедалі більше часу. Вона забезпечує масштабованість процесу перевірки без необхідності пропорційного збільшення людських ресурсів, що робить її економічно вигідною та стратегічно важливою [15].

Автоматизоване тестування охоплює різні рівні: від модульного, яке перевіряє окремі функції чи компоненти системи, до інтеграційного, що аналізує взаємодію між ними. Далі йде системне тестування, яке оцінює роботу всієї програми як єдиного цілого, та приймальне тестування, що визначає готовність продукту для передачі замовнику. Окрім того, автоматизовані методи активно застосовуються в функціональному тестуванні — для перевірки відповідності логіки вимогам, а також у нефункціональному — зокрема для оцінки продуктивності, безпеки, доступності чи зручності.

Таким чином, автоматизоване тестування виходить за межі технічної необхідності, стаючи стратегічним елементом сучасного процесу розробки, який сприяє підвищенню якості, надійності та конкурентоспроможності програмного забезпечення.

2.2 Види автоматизованого тестування

Класифікація видів автоматизованого тестування допомагає визначити, які характеристики програмного забезпечення вигідно перевіряти за допомогою автоматизованих інструментів. На відміну від загальної класифікації тестування,

що охоплює як ручні, так і автоматизовані методи для оцінки функціональних і нефункціональних властивостей системи, класифікація автоматизованого тестування зосереджується на тих аспектах, де автоматизація є технічно здійсненною, економічно виправданою та сприяє підвищенню якості.

Автоматизоване тестування найбільше виправдане в ситуаціях, коли перевірки проводяться регулярно, у великих обсягах або вимагають високої точності та повторюваності. Зокрема, до таких належить функціональне тестування, яке передбачає перевірку відповідності системи вимогам через заздалегідь підготовлені сценарії, реалізовані у вигляді скриптів. Автоматизація в цьому випадку значно скорочує час, необхідний для тестування численних функціональних компонентів.

Значного поширення набуло автоматизоване регресійне тестування, яке полягає в повторному виконанні наявних тестів після внесення змін до коду. Цей метод дозволяє швидко виявляти повторне виникнення виправлених раніше помилок або появу нових дефектів, пов’язаних із оновленнями функціоналу. У цьому контексті автоматизація забезпечує безперервний контроль стабільності системи під час її еволюції.

Окрему категорію становить автоматизоване тестування продуктивності, яке включає навантажувальне, стресове та об’ємне тестування. Ці види складно реалізувати вручну через необхідність обробки великої кількості одночасних запитів, тривалих сценаріїв або моделювання пікових навантажень. Використання спеціалізованих інструментів не лише автоматизує процес, а й забезпечує точні показники продуктивності системи в реальному часі.

Важливе значення має також автоматизоване інтеграційне тестування, яке контролює правильність взаємодії між компонентами системи. Воно є особливо актуальним для мікросервісної архітектури чи складних модульних рішень, де автоматизовані перевірки допомагають оперативно виявляти збої на рівні комунікації між частинами програмного продукту.

Ще одним прикладом є автоматизоване тестування API, яке дозволяє перевіряти програмні інтерфейси без залучення користувацького інтерфейсу. Такий

метод забезпечує високу швидкість виконання тестів і дає змогу оцінити критичну логіку бекенд-систем ще до впровадження фронтенд-компонентів [1,2].

Отже, види автоматизованого тестування не витісняють, а вдосконалюють загальні методики тестування, акцентуючи увагу на тих напрямках, де автоматизація гарантує максимальну ефективність. У динамічному циклі розробки такі підходи стають ключовим інструментом для оптимізації якості програмного продукту.

2.3 Методи автоматизованого тестування

Методи автоматизованого тестування вирішують питання: «Як організувати процес автоматичної перевірки програмного забезпечення?». Їхня головна мета — визначити підхід до реалізації тестування за допомогою спеціалізованих інструментів і скриптів, які зменшують залучення людини до виконання рутинних перевірок.

На відміну від загальних методів тестування, що класифікуються переважно за доступом до коду чи етапами (наприклад, метод «чорного ящика» чи модульне тестування), методи автоматизованого тестування акцентують увагу на рівні та способі автоматизації. Вони стосуються технічної організації процесу, вибору стратегій для структурування тестів, взаємодії з системою та можливості повторного використання сценаріїв.

Основна класифікація методів автоматизованого тестування включає такі підходи:

• Скриптове тестування (scripted testing) — створення тестових сценаріїв у вигляді чітко визначених скриптів із заданою послідовністю дій і прогнозованими результатами. Цей метод гарантує високу точність виконання та зручність для повторного застосування, особливо в регресійному тестуванні.

• Тестування на основі ключових слів (keyword-driven testing) — розробка тестів із використанням наборів ключових команд або дій, що застосовуються до

елементів інтерфейсу. Такий підхід відокремлює логіку тестування від її реалізації, роблячи тести доступними навіть для фахівців із мінімальними технічними знаннями.

• Тестування, кероване даними (data-driven testing) — повторне виконання одного тестового сценарію з різними наборами вхідних даних. Це підвищує гнучкість і дає змогу ефективно перевіряти реакцію системи на різні комбінації параметрів.

• Модельно-орієнтоване тестування (model-based testing) — автоматичне генерування тестових сценаріїв на основі моделі поведінки системи. Цей метод особливо корисний для великих систем із складною логікою, де ручне створення тестів виявляється малопродуктивним.

Отже, методи автоматизованого тестування зосереджуються не стільки на об’єктах тестування (як у загальних методах), скільки на способах автоматизації процесу. Вони створюють основу для побудови стабільного, масштабованого та відтворюваного тестового процесу, сприяючи ефективному виявленню дефектів на різних етапах життєвого циклу розробки [2].

2.4 Типи автоматизованого тестування

Типи автоматизованого тестування визначаються залежно від рівня перевірки програмного забезпечення, а також від цілей і масштабів тестування. Вони допомагають визначити, які етапи розробки та функціональні елементи системи доцільно й ефективно перевіряти за допомогою автоматизованих засобів.

Автоматизоване тестування охоплює кілька ключових типів, які забезпечують системний підхід до підтримання якості програмного продукту на різних етапах його життєвого циклу. Одним із них є модульне тестування, яке зосереджується на перевірці окремих компонентів або функціональних блоків програми. Автоматизація на цьому етапі дозволяє швидко виявляти логічні помилки в модулях ще до їхньої інтеграції в ширші системні структури.

Інтеграційне тестування в автоматизованому форматі спрямоване на контроль правильності взаємодії між окремими модулями чи підсистемами. Цей метод допомагає виявляти проблеми сумісності, збої в обміні даними чи логічні неточності, які можуть виникати під час об’єднання компонентів.

На рівні системного тестування автоматизація передбачає всебічну перевірку цілісної програми, часто в умовах, максимально наближених до реального середовища використання. Це дає змогу оцінити не лише функціональні можливості, а й нефункціональні характеристики, такі як продуктивність, стабільність та надійність.

Регресійне тестування, важлива складова забезпечення якості програмного забезпечення, автоматизується для повторного виконання вже розроблених тестів після змін у коді. Такий підхід дозволяє вчасно виявляти помилки, спричинені модифікаціями, і підтримує стабільність системи.

Навантажувальне автоматизоване тестування зосереджується на аналізі поведінки системи під різними навантаженнями. За допомогою спеціалізованих інструментів і сценаріїв моделюються реальні умови експлуатації, що сприяє виявленню проблем із масштабуванням та слабких місць у продуктивності.

Таким чином, типи автоматизованого тестування створюють цілісну структуру, яка забезпечує послідовне й комплексне підвищення якості програмного продукту, оптимізуючи процеси контролю та підвищуючи ефективність тестування на кожному етапі розробки [1].

2.5 Інструменти для функціонального тестування

Функціональне тестування є фундаментальним і одним із ключових напрямів забезпечення якості вебдодатків. Його основна мета — перевірити, чи відповідає реальна поведінка системи її функціональним вимогам. Іншими словами, цей вид тестування зосереджується на тому, що система виконує, а не на способі реалізації цих дій, прагнучи підтвердити коректність роботи кожної функції відповідно до специфікацій чи очікувань користувачів.

У контексті вебдодатків функціональне тестування зазвичай включає валідацію логіки бізнес-процесів, контроль обробки введених користувачем даних, коректність виконання серверних запитів, відображення контенту та перевірку інтеграцій із внутрішніми модулями чи зовнішніми системами. До цього можуть належати такі етапи, як реєстрація користувача, авторизація, переміщення між сторінками, заповнення форм, обробка помилок і відповідні повідомлення.

Ці тести можуть проводитися як вручну, так і з використанням автоматизованих інструментів. У невеликих проєктах ручне тестування залишається практичним рішенням, але з ускладненням системи та зростанням кількості функцій виникає необхідність у автоматизації. Автоматизоване функціональне тестування економить час, гарантує повторюваність перевірок і зменшує ймовірність помилок через людський фактор.

Зазвичай функціональне тестування реалізується на рівнях модульного, інтеграційного, системного чи приймального тестування. Кожен із цих рівнів відповідає за свою частину перевірки, забезпечуючи всебічне охоплення функціональності. У сучасному програмному забезпеченні також активно застосовується тестування API, яке є важливою ланкою функціонального тестування в багаторівневих вебдодатках.

Ефективне функціональне тестування відіграє ключову роль у зниженні кількості критичних дефектів у кінцевій версії програмного продукту, забезпечуючи стабільну роботу та покращуючи взаємодію користувачів із системою. Воно сприяє підвищенню загальної надійності програмного забезпечення, що є критично важливим для створення якісного та довіреного продукту. Значущість функціонального тестування особливо зростає в умовах динамічної розробки, де функціонал системи регулярно оновлюється, розширюється та адаптується до нових вимог. Завдяки ретельній перевірці кожної функції вдається уникнути помилок, які можуть негативно вплинути на користувацький досвід, та забезпечити відповідність продукту очікуванням клієнтів. Таким чином, функціональне тестування стає невід’ємною частиною процесу розробки, що гарантує високу якість і конкурентоспроможність програмного забезпечення.

2.5.1 Selenium

Selenium є одним із найстаріших і найвпізнаваніших фреймворків для автоматизованого тестування вебдодатків. Він підтримує широкий спектр мов програмування (Java, Python, C#, JavaScript тощо) та браузерів (Chrome, Firefox, Edge, Safari), що робить його гнучким і універсальним інструментом для тестування.

Основним елементом є Selenium WebDriver, який дає змогу імітувати дії користувача в браузері, що особливо цінно для функціонального та кросбраузерного тестування. Цей фреймворк активно інтегрується в процеси CI/CD і підтримується завдяки великій спільноті, яка сприяє його розвитку та оновленню.

Водночас Selenium має як сильні сторони, так і певні обмеження, які варто враховувати при виборі інструменту для автоматизації. Серед переваг — підтримка багатьох мов програмування, таких як Java, Python, C#, Ruby, JavaScript, що полегшує інтеграцію в проєкти з різними технологіями та дозволяє командам обирати зручний стек. Іншою значною перевагою є кросбраузерність: Selenium сумісний із провідними браузерами (Chrome, Firefox, Edge, Safari), забезпечуючи широке охоплення середовищ роботи вебзастосунків. Крім того, його легка інтеграція з платформами CI/CD сприяє автоматизації розгортання та тестування, підтримує стабільність і якість на всіх етапах розробки. Популярність фреймворку значною мірою зумовлена активною спільнотою, яка регулярно оновлює документацію, ділиться прикладами, вирішує технічні питання та вдосконалює бібліотеки [16].

Проте Selenium має й недоліки. Одним із головних є відносно високий поріг входу для новачків: робота з ним вимагає не лише знань обраної мови програмування, а й розуміння принципів роботи вебінтерфейсів, структури DOM та синхронізації елементів. Крім того, фреймворк не має вбудованих засобів для генерації звітів про тести, що змушує розробників звертатися до додаткових бібліотек або створювати власні інструменти. Ще однією складністю є тестування динамічних елементів, які змінюються під час завантаження сторінки чи взаємодії

з нею — тут потрібне точне налаштування затримок (waits) і обробка винятків. Порівняно з сучасними фреймворками, такими як Cypress чи Playwright, інтерфейс Selenium може здаватися менш інтуїтивним, що ускладнює створення автотестів, особливо на початкових етапах.

Selenium добре підходить для великих проєктів, де потрібен повний контроль над тестуванням і широке охоплення браузерів [3,4].

2.5.2 Cypress

Cypress — це сучасний інструмент для автоматизованого тестування вебдодатків, який відзначається простотою у використанні та високою ефективністю. Розроблений на базі JavaScript, він тісно інтегрується з фронтенд-стеком, що робить його зручним для перевірки сучасних вебзастосунків.

Цей інструмент виконує тести безпосередньо в браузері, дозволяючи точно відстежувати поведінку додатку в реальних умовах. Завдяки цьому тести проходять швидко та стабільно. Cypress пропонує зручний інтерфейс для налагодження, автоматично створює скріншоти та записує відео тестів, полегшуючи аналіз результатів.

Серед ключових переваг Cypress — його легка інтеграція з проєктами на JavaScript, що особливо вигідно для розробників, які працюють із сучасними фреймворками, такими як React, Angular чи Vue. Інструмент забезпечує високу швидкість виконання тестів, значно прискорюючи контроль якості коду на різних етапах розробки. До того ж, інтуїтивний інтерфейс для налагодження спрощує пошук помилок і відтворення тестів. Важливою перевагою є також детальна, регулярно оновлювана документація з численними прикладами, яка полегшує освоєння інструменту як новачкам, так і досвідченим спеціалістам [16].

Разом із тим, у Cypress є певні обмеження, які варто враховувати при виборі. Зокрема, він підтримує лише JavaScript і TypeScript, що звужує його застосування в проєктах на інших мовах програмування. Крім того, обмежена кросбраузерність — відсутність підтримки Safari — може ускладнити охоплення всіх

тестувальних середовищ. Окрім того, Cypress орієнтований виключно на вебдодатки й не підтримує тестування мобільних браузерів чи нативних мобільних програм. Ще одним нюансом є те, що тести проводяться в контрольованому середовищі інструменту, що іноді може призводити до розбіжностей із реальними умовами використання [18].

Cypress найкраще підходить для фронтенд-команд, які потребують швидкого зворотного зв'язку та зручного інструменту для створення тестів [5].

2.5.3 Playwright

Playwright — це інструмент від Microsoft, який пропонує широкі можливості для автоматизованого тестування вебдодатків. Він підтримує мови програмування, такі як JavaScript, Python, C# і Java, а також дозволяє проводити тести в основних браузерах (Chromium, Firefox, WebKit), зокрема й у Safari.

Цей інструмент вирізняється наявністю численних функцій «з коробки»: тестування в кількох вкладках, імітація роботи на мобільних пристроях, запис відео, створення знімків екрана та навіть підтримка тестування в офлайн-режимі. Його гнучкість дозволяє ефективно працювати з динамічними інтерфейсами та асинхронними подіями без потреби в додаткових бібліотеках.

Playwright має ряд переваг, які виділяють його серед подібних рішень. Серед ключових — підтримка кількох мов програмування, включаючи JavaScript, TypeScript, Python, Java та C#, що забезпечує гнучкість інтеграції в різноманітні проєкти. Важливою рисою є повна кросбраузерність, яка охоплює не лише популярні браузери, як-от Chrome і Firefox, а й Safari, що гарантує охоплення широкого спектра користувацьких сценаріїв. Playwright стабільно справляється з динамічними вебінтерфейсами, зокрема з елементами, що змінюються під час взаємодії чи завантаження, завдяки вбудованим механізмам очікування та відстеження стану. Також інструмент пропонує функцію імітації мобільних пристроїв, що дозволяє тестувати адаптивність вебдодатків без фізичного обладнання [16].

Разом із тим, Playwright має певні обмеження. Як відносно новий фреймворк, він поступається за кількістю навчальних матеріалів, посібників і прикладів порівняно з більш усталеними інструментами, такими як Selenium, що може ускладнити освоєння для новачків або команд, які тільки починають його використовувати. Для реалізації складних тестувальних сценаріїв часто потрібні додаткові налаштування, що вимагає глибшого розуміння внутрішньої роботи інструменту. Ефективне застосування Playwright також потребує високого рівня технічної підготовки, зокрема знань програмування, роботи з асинхронними операціями та налагодження тестового середовища.

Playwright стане ідеальним рішенням для команд, які прагнуть детального контролю над тестуванням і роботи з передовими вебтехнологіями [6].

2.5.4 Postman

Postman — один із найпопулярніших інструментів для автоматизованого тестування API, який широко застосовується під час розробки вебдодатків. Його основна перевага полягає в орієнтації на тестування RESTful API, що робить його незамінним для оцінки взаємодії між клієнтом і сервером. Інструмент пропонує інтуїтивно зрозумілий інтерфейс, який дозволяє створювати, відправляти та автоматизувати перевірку HTTP-запитів без потреби в написанні складного коду на початкових етапах.

Серед ключових сильних сторін Postman — підтримка JavaScript для створення тестових сценаріїв. Це дає змогу автоматизувати перевірку відповідей API, зокрема валідацію статус-кодів, структури JSON-об’єктів, часу виконання запитів, авторизації та заголовків. Крім того, інструмент підтримує змінні середовища, колекції запитів, логіку виконання тестів і функцію збереження та повторного використання запитів, що полегшує автоматизацію складних тестових сценаріїв.

Postman інтегрується з утилітою командного рядка Newman, яка дозволяє запускати колекції запитів у автоматичному режимі без графічного інтерфейсу.

Це особливо корисно в рамках CI/CD-процесів, де тести необхідно виконувати автоматично після оновлення коду чи зміни конфігурації. Newman сумісний із провідними CI-системами (Jenkins, GitLab CI, GitHub Actions, Azure DevOps), що сприяє безперешкодному впровадженню автоматизованого тестування API в загальний цикл розробки [18].

До інших переваг Postman належать підтримка різних механізмів автентифікації (API Keys, OAuth 2.0, Bearer Tokens), можливість генерування документації API на основі колекцій, а також зручна візуалізація відповідей. Інструмент придатний як для одноразових перевірок, так і для масштабних автоматизованих наборів тестів, які охоплюють усі аспекти роботи API [17].

Водночас Postman має певні обмеження. Він зосереджений виключно на рівні API й не підтримує повноцінне тестування користувацького інтерфейсу, що ускладнює одночасну перевірку інтегрованої поведінки UI та API без додаткових інструментів. Хоча створення простих тестів є доступним, для складніших перевірок потрібні знання JavaScript і розуміння структури API. Крім того, обмежена підтримка складних сценаріїв із умовними переходами чи циклічними залежностями може вимагати використання зовнішніх інструментів або розширень через інтеграції.

Postman стане оптимальним вибором для команд, які активно працюють із API, прагнуть прискорити оцінку серверної логіки, забезпечити стабільність обміну даними та інтегрувати тестування в процес розгортання. Завдяки своїй простоті, доступності та активній спільноті, він залишається одним із провідних інструментів у сфері сучасного автоматизованого тестування [6].

2.6 Інструменти для тестування продуктивності

Тестування продуктивності (performance testing) відіграє ключову роль у забезпеченні якості вебдодатків, особливо тих, які призначені для обслуговування великої кількості користувачів або обробки значних обсягів даних. Його

основна мета — оцінити поведінку додатку під навантаженням, визначити швидкість реагування на запити та перевірити стабільність роботи в різних умовах.

Застосування спеціалізованих інструментів для тестування продуктивності дає змогу імітувати тисячі одночасних користувачів, виявляти слабкі місця в архітектурі, аналізувати час відгуку серверів, пропускну здатність і використання ресурсів. Такий підхід допомагає підготувати систему до реальних навантажень і запобігти збоїв після випуску.

Серед найвідоміших інструментів у цій галузі — Apache JMeter, k6 та Locust. Кожен із них вирізняється унікальними особливостями, перевагами та сферами застосування, які залежать від технічних вимог і специфіки проєкту.

У наступних розділах буде проведено детальніший огляд цих інструментів із висвітленням їхніх сильних і слабких сторін.

2.6.1 Apache JMeter

Apache JMeter — один із найстаріших і найбільш авторитетних інструментів для тестування продуктивності вебдодатків та інших серверних служб. Розроблений під егідою Apache Software Foundation, цей інструмент є повністю безкоштовним і має відкритий вихідний код, що сприяє його популярності серед компаній і тестувальників по всьому світу.

JMeter дозволяє симулювати навантаження на вебсервери, бази даних, FTP-сервери, REST/SOAP API та інші сервіси, створюючи сотні чи тисячі віртуальних користувачів. Завдяки зручному графічному інтерфейсу та підтримці скриптування й інтеграції з CI/CD-системами, такими як Jenkins, він забезпечує автоматизоване навантажувальне тестування протягом усього процесу розробки.

Apache JMeter є одним із лідерів у сфері навантажувального тестування завдяки низці значних переваг. Серед них — підтримка широкого спектра протоколів, включаючи HTTP, FTP, JDBC, SOAP, REST, TCP, SMTP та інші, що робить його придатним для тестування не лише вебдодатків, а й баз даних, пошто-

вих серверів, вебсервісів та інших платформ. Важливою рисою є можливість розподіленого тестування, коли навантаження розподіляється між кількома машинами, імітуючи реальні сценарії з великою кількістю одночасних користувачів. Гнучкість конфігурації, забезпечена інтуїтивним графічним інтерфейсом, дозволяє створювати складні тестові сценарії без програмування. Активна спільнота користувачів, регулярні оновлення та розширений набір плагінів і документації сприяють його вдосконаленню. Крім того, JMeter легко інтегрується з платформами для збору метрик і CI/CD, підтримуючи DevOps-підхід до автоматизації тестування.

Разом із тим, JMeter має певні недоліки, які можуть впливати на його ефективність у деяких проєктах. Зокрема, графічний інтерфейс характеризується високим споживанням ресурсів, що може знижувати продуктивність на слабших комп'ютерах. Окрім того, велика кількість налаштувань і можливостей може ускладнити освоєння інструменту для новачків, особливо тих, хто не має досвіду тестування продуктивності. Хоча JMeter підтримує скриптове керування, його архітектура менш зручна для повністю автоматизованого тестування порівняно з сучасними рішеннями, такими як k6, які спочатку розроблялися з урахуванням кодової автоматизації та інтеграції в CI/CD-середовище.

Завдяки своїй універсальності та стабільності, Apache JMeter залишається еталонним рішенням для навантажувального тестування, особливо в корпоративному секторі, де потрібні висока масштабованість і гнучкість [7].

2.6.2. Locust

Locust — це популярний інструмент для тестування продуктивності, який вирізняється своєю простотою та значною гнучкістю. Його головна особливість полягає в застосуванні мови програмування Python для створення сценаріїв навантажувального тестування, що робить його привабливим для інженерів, які вже володіють цією мовою.

Locust дає змогу імітувати поведінку великої кількості користувачів, які одночасно взаємодіють із вебдодатком, і відстежувати, як система витримує навантаження. На відміну від традиційних інструментів, він пропонує вебінтерфейс у реальному часі, що дозволяє спостерігати за результатами тестів безпосередньо під час їх виконання.

Locust здобув широке визнання як інструмент для навантажувального тестування завдяки своїй простоті, гнучкості та орієнтації на Python. Однією з головних переваг є можливість писати тестові сценарії з використанням Python, що зручно для розробників, знайомих із цією мовою, і дозволяє створювати складні сценарії з довільною логікою поведінки, включаючи змінні, умовні оператори, цикли та зовнішні бібліотеки. Інструмент також забезпечує зручний вебінтерфейс для відстеження результатів у реальному часі, де доступні графіки навантаження, час відгуку та статистика помилок. Важливою рисою є масштабованість: Locust підтримує розподілене тестування, дозволяючи запускати тисячі віртуальних користувачів на кількох хостах одночасно. Крім того, його простота налаштування та сумісність із засобами автоматизації сприяють інтеграції в CI/CD-процеси, що полегшує регулярне тестування продуктивності під час розробки.

Разом із тим, Locust має певні обмеження. Найпомітнішим є його орієнтація переважно на HTTP-протокол, що звужує можливості тестування інших чи складніших протоколів, таких як WebSocket або FTP. Окрім того, хоча знання Python є перевагою для досвідчених користувачів, для тих, хто не володіє цією мовою, робота з інструментом може виявитися менш інтуїтивною. Ще одним викликом є необхідність ручного налаштування інфраструктури для тестів із високим навантаженням, наприклад, конфігурація кількох worker-хостів, що вимагає додаткових навичок адміністрування та роботи з мережами.

Locust є особливо зручним для проєктів, де вже активно використовується Python, або коли потрібна висока гнучкість у моделюванні поведінки користувача. Його легкість та інтерактивність роблять його популярним вибором для навантажувального тестування у багатьох сучасних вебпроєктах [8].

2.6.3. k6

k6 — сучасний інструмент для тестування продуктивності від Grafana Labs, призначений для розробників і DevOps-команд. Він дозволяє створювати скрипти навантажувального тестування на JavaScript, забезпечуючи гнучкість і просту інтеграцію в CI/CD-процеси. На відміну від JMeter із графічним інтерфейсом, k6 є консольним інструментом, орієнтованим на автоматизацію, що робить його легким, швидким і зручним для хмарних середовищ, таких як Kubernetes чи GitHub Actions. k6 підтримує стрес-тестування, перевірку витривалості систем і вимірювання показників, як-от час відповіді та кількість запитів на секунду.

Завдяки JavaScript як мові скриптів, k6 особливо зручний для веброзробників, адже створення та підтримка тестів є інтуїтивними й не вимагають значного навчання. Інструмент вирізняється високою продуктивністю, масштабованістю та легкістю, дозволяючи обробляти великі навантаження навіть на обмежених ресурсах. Він підтримує сучасні інтерфейси, як-от REST API, WebSocket і GraphQL, а чітка структура сценаріїв сприяє зрозумілості та повторному використанню коду.

Однак k6 має й недоліки. Відсутність графічного інтерфейсу може ускладнити роботу новачкам, оскільки налаштування й запуск тестів виконуються через командний рядок. Інструмент підтримує обмежену кількість протоколів, не включаючи SOAP, FTP чи SMTP, що звужує його застосування порівняно з універсальнішими рішеннями, як JMeter. Також деякі розширені функції, зокрема детальна аналітика та візуалізація в реальному часі, доступні лише в платній версії k6 Cloud, що може бути обмеженням для команд із невеликим бюджетом.

Загалом, завдяки сучасній архітектурі, простому синтаксису та високій продуктивності, k6 є перспективним рішенням для навантажувального тестування в DevOps-орієнтованих проєктах [9].

2.7 Інструменти для тестування безпеки

Безпека вебдодатків відіграє ключову роль у забезпеченні якості програмного забезпечення, особливо з огляду на постійне зростання кіберзагроз. Навіть незначні вразливості можуть спричинити серйозні наслідки, такі як витік конфіденційних даних, фінансові збитки чи втрата репутації компанії. Тому тестування безпеки є обов'язковою частиною процесу створення та впровадження вебдодатків.

Для гарантії належного рівня захисту застосовуються спеціалізовані інструменти, які допомагають виявляти вразливості на ранніх етапах розробки. Одним із найпоширеніших і доступних інструментів є OWASP ZAP (Zed Attack Proxy).

OWASP ZAP — це безкоштовний інструмент із відкритим вихідним кодом для автоматизованого тестування безпеки вебдодатків, розроблений у рамках проєкту OWASP (Open Worldwide Application Security Project). Він є одним із провідних рішень для динамічного аналізу безпеки (DAST) і широко застосовується як для ручного, так і для автоматизованого тестування. ZAP імітує потенційні атаки на вебдодатки, щоб виявити вразливості, які можуть бути використані зловмисниками. До них належать SQL-ін'єкції, міжсайтовий скриптинг (XSS), помилки в управлінні сесіями, слабкі механізми автентифікації та витоки конфіденційних даних через неналежні налаштування сервера чи недостатнє шифрування. Ці можливості роблять ZAP цінним інструментом для раннього виявлення та усунення критичних ризиків безпеки.

Інструмент оснащений зручним графічним інтерфейсом і підтримує інтеграцію з CI/CD-системами, такими як Jenkins, Docker і GitHub Actions, що робить його корисним для DevOps-команд і фахівців із кібербезпеки. Відкритий код і активна спільнота забезпечують регулярні оновлення та підтримку, а можливість створення власних правил сканування дозволяє адаптувати ZAP до специфічних потреб проєкту. Завдяки доступності та простоті освоєння, він підходить як для освітніх, так і для комерційних цілей.

Проте ZAP має й обмеження. Наприклад, він може генерувати хибнопозитивні результати, що вимагає додаткового аналізу. Підтримка складних API, таких як GraphQL, обмежена без спеціальних плагінів або ручного налаштування. Крім того, ефективне використання ручного режиму потребує знань у сфері кібербезпеки, зокрема розуміння вебпротоколів і методів експлуатації вразливостей.

Загалом OWASP ZAP є потужним і доступним інструментом, який ідеально підходить як для новачків, так і для досвідчених тестувальників. Його широке застосування в індустрії підкреслює важливість інтеграції тестування безпеки в процес розробки вебдодатків [10].

2.8 Інструменти для інтеграції тестування в CI/CD

У сучасній практиці розробки програмного забезпечення надзвичайно важливими стали підходи, які дозволяють забезпечити постійний контроль якості продукту під час активного розвитку проєкту. Серед таких підходів особливе місце займають принципи безперервної інтеграції (Continuous Integration, CI) та безперервної доставки (Continuous Delivery, CD). Вони передбачають тісне поєднання процесів написання коду, його тестування та впровадження в робоче середовище. Кожна нова зміна, яку розробник вносить до коду, одразу ж потрапляє у спільне середовище, де автоматично перевіряється на коректність. Таким чином забезпечується швидке виявлення помилок і зменшується ймовірність того, що дефекти потраплять у фінальну версію продукту.

У рамках цих підходів усі етапи перевірки — від запуску тестів до отримання результатів — автоматизуються. Це означає, що розробнику або тестувальнику не потрібно кожного разу вручну запускати тести, слідкувати за відповідністю результатів чи налаштовувати середовище — усе відбувається автоматично після кожного оновлення коду. Завдяки цьому команда може зосередитись на розробці нового функціоналу, не витрачаючи час на рутинні перевірки. Крім

того, автоматичний запуск тестів при кожному внесенні змін дозволяє дуже швидко виявити, яка саме частина коду спричинила збій, і відкотити її або виправити до того, як проблема стане критичною.

Ще однією важливою перевагою є прискорення випуску нових версій продукту. Якщо система CI/CD налаштована правильно, то після успішного проходження всіх тестів зміни можуть автоматично передаватись на тестові або навіть продуктивні сервери. Це значно скорочує цикл релізу: те, що раніше могло займати дні або тижні, тепер відбувається за лічені години або навіть хвилини. В умовах високої конкуренції на ринку програмних продуктів це дає компаніям значну перевагу, адже дозволяє швидко реагувати на потреби користувачів і впроваджувати нові функції чи виправлення.

Для реалізації CI/CD-процесів активно використовуються спеціальні інструменти, які інтегруються в інфраструктуру проєкту та забезпечують стабільне й контрольоване виконання всіх необхідних дій. Найчастіше до таких інструментів належать системи автоматизації процесів (наприклад, Jenkins) та контейнеризації (як-от Docker). У межах цього підрозділу не розглядається їхній детальний опис, однак важливо підкреслити, що саме вони дозволяють зберігати стабільність, повторюваність та ефективність автоматизованого тестування в рамках CI/CD.

2.8.1. Jenkins

Jenkins — це широко використовуваний інструмент із відкритим кодом для автоматизації безперервної інтеграції та доставки. Він дозволяє створювати пайплайни, які включають послідовні етапи: компіляцію коду, виконання автоматизованих тестів, аналіз покриття коду, створення артефактів і розгортання.

Jenkins є одним із провідних рішень для автоматизації розробки, тестування та деплою програмного забезпечення, відіграючи центральну роль у реалізації CI/CD. Його ключова перевага — гнучка система плагінів, що забезпечує інтеграцію з інструментами тестування (Selenium, JMeter, Cypress, OWASP ZAP),

засобами аналізу якості коду та моніторингу. Це робить Jenkins адаптивним для різних етапів розробки та придатним як для невеликих команд, так і для масштабних корпоративних проєктів.

Ще одна важлива особливість — можливість створення складних сценаріїв автоматизації через Jenkinsfile, файл із декларативним або скриптовим синтаксисом, який чітко визначає логіку компіляції, тестування та розгортання. Jenkins підтримує популярні системи контролю версій, зокрема Git, що спрощує відстеження змін і автоматичний запуск пайплайнів після комітів. Активна спільнота, обширна документація та навчальні ресурси полегшують освоєння інструмента й вирішення технічних питань. Jenkins також добре інтегрується з DevOps-інструментами, такими як Docker, Kubernetes і Ansible, дозволяючи створювати масштабовані та надійні пайплайни.

Проте Jenkins має й недоліки. Налаштування та адміністрування можуть бути складними, особливо в масштабних середовищах. Для стабільної роботи серверів Jenkins при великій кількості проєктів і частих запусках пайплайнів потрібні значні ресурси та зусилля для моніторингу й оптимізації. Інтерфейс інструмента може здаватися громіздким і не завжди інтуїтивним для новачків. У великих проєктах Jenkins може споживати багато ресурсів (пам’яті та процесора), що потребує ретельного планування інфраструктури.

Завдяки Jenkins можна створити автоматизоване середовище, де тестування запускається після кожної зміни коду, значно знижуючи ризик помилок на пізніх етапах розробки [11].

2.8.2. Docker

Docker — це платформа контейнеризації, яка забезпечує створення ізольованих середовищ для запуску програм і тестів. У сфері автоматизованого тестування Docker застосовується для формування однакових середовищ, незалежних від операційної системи розробника чи сервера.

Docker є важливим елементом сучасної DevOps-екосистеми та активно використовується для контейнеризації додатків. Його основна перевага полягає у створенні стандартизованих, ізольованих середовищ, які залишаються ідентичними на етапах розробки, тестування та продакшену. Це усуває проблему "працює на моїй машині", мінімізуючи невідповідності між різними системами.

Контейнери Docker дозволяють швидко розгортати додатки та сервіси, що є особливо цінним для команд, які застосовують принципи безперервної інтеграції та доставки. Інструмент підтримує як вертикальне, так і горизонтальне масштабування, що сприяє ефективному управлінню навантаженням і адаптації до змін вимог продуктивності. Docker також легко інтегрується з інструментами автоматизації, такими як Jenkins, GitLab CI, GitHub Actions, що спрощує створення CI/CD-процесів із мінімальними затратами на налаштування середовища.

Попри численні переваги, Docker має певні недоліки. Для його ефективного використання необхідно опанувати нові концепції та практики, що може вимагати додаткового навчання, особливо для тих, хто не мав досвіду роботи з віртуалізацією чи інфраструктурою. Налаштування конфігураційних файлів, таких як Dockerfile і docker-compose.yml, потребує технічних знань і уваги до деталей. Крім того, контейнеризація може бути обмеженою для додатків, які потребують тісної інтеграції з операційною системою чи спеціального доступу до обладнання, що може ускладнити або унеможливити їх використання без значних змін в архітектурі.

Docker забезпечує тестування в стандартизованих умовах, гарантуючи стабільність результатів незалежно від середовища. У поєднанні з Jenkins він створює потужну основу для CI/CD-практик, що відповідає потребам сучасної розробки [12].

3 ТЕСТУВАННЯ WEB ІНТЕРФЕЙСУ ТА API СИСТЕМИ УПРАВЛІННЯ ЛІНГВІСТИЧНИМИ ЗАДАЧАМИ LINGUIST PORTAL

У сучасних умовах стрімкого розвитку інформаційних технологій автоматизоване тестування стало невід'ємною складовою процесу створення програмного забезпечення, оскільки саме воно дозволяє підтримувати високий рівень якості, забезпечувати відповідність функціональності очікуванням користувачів та оперативно виявляти дефекти на різних етапах життєвого циклу продукту. У складних і динамічних програмних системах, де реалізуються численні інтерактивні сценарії, регулярно відбуваються зміни функціоналу, а вимоги користувачів постійно оновлюються, автоматизація стає не лише корисним доповненням, а фактично необхідністю для ефективного контролю якості.

Для системи Linguist Portal, яка призначена для управління лінгвістичними задачами, автоматизоване тестування виконує особливо важливу роль. Ця система містить складну логіку обробки даних, взаємодію з великою кількістю користувачів, підтримку різних мовних пар та специфікацій, що вимагає постійної перевірки коректності її роботи. У таких умовах автоматизоване тестування стає ключовим інструментом для забезпечення стабільності роботи, цілісності бізнес-логіки та зручності користування інтерфейсом. Воно дозволяє вчасно виявляти як технічні помилки, так і недоліки в логіці побудови процесів, знижуючи ризик виникнення критичних помилок у продуктивному середовищі.

Цей розділ присвячено детальному аналізу процесу автоматизованого тестування додатку Linguist Portal. У ньому розглядається загальна структура системи з точки зору тестування, визначаються основні об'єкти тестування, тобто елементи web інтерфейсу та API, які підлягають перевірці. Особливу увагу приділено обґрунтуванню вибору інструментів тестування, таких як Cypress для UI-тестів та Postman для перевірки API-запитів, що дозволяє охопити всі ключові аспекти роботи системи. Також у межах цього аналізу описуються методи тестування, які були застосовані, включно з функціональними, регресійними та нефункціональними перевірками. У результаті буде сформовано комплексне уявлення

про роль і ефективність автоматизованого тестування у підтримці якості програмного продукту Linguist Portal.

3.1 Опис додатку Linguist Portal, його структура та об’єкти тестування

Linguist Portal — це інноваційна платформа, розроблена для автоматизації процесів обробки лінгвістичних даних [14]. Вона створена для підтримки лінгвістів, перекладачів, редакторів і менеджерів проєктів, які працюють із текстовими даними. Основна мета системи — спростити управління завданнями, пов’язаними з перекладом, локалізацією, редагуванням текстів і формуванням звітів, що підвищує ефективність роботи команд. Завдяки застосуванню сучасних технологій, Linguist Portal забезпечує зручну взаємодію користувачів із лінгвістичними ресурсами, оптимізуючи робочі процеси.

Система складається з двох основних компонентів, які тісно взаємодіють для забезпечення повноцінної роботи платформи. Web інтерфейс є головним каналом доступу користувачів до системи, дозволяючи створювати, редагувати та керувати лінгвістичними задачами, переглядати звіти та взаємодіяти з іншими учасниками проєктів. Він побудований на основі технології ASP.NET, яка забезпечує надійність, безпеку та ефективну обробку серверних запитів. Web інтерфейс використовує HTML, CSS і JavaScript для створення динамічних елементів, а серверна логіка, реалізована через ASP.NET, гарантує стабільну роботу навіть за високих навантажень. API, своєю чергою, виконує роль серверної частини, обробляючи запити від web інтерфейсу та зовнішніх клієнтів. Воно реалізовано за принципами REST, що забезпечує стандартизований обмін даними у форматі JSON, дозволяючи створювати, оновлювати, видаляти та отримувати інформацію про задачі, а також інтегруватися з іншими системами.

Тестування Linguist Portal поділено на два ключові напрямки для забезпечення комплексного контролю якості. Тестування web інтерфейсу зосереджено

на перевірці коректності відображення елементів, зручності навігації, правильності роботи форм, кнопок, фільтрів і таблиць. Особлива увага приділяється перевірці авторизації, створення та редагування задач, а також відображення звітів. Тестування API спрямоване на перевірку обробки запитів, відповідності формату даних, правильності статус-кодів і стабільності роботи під різними навантаженнями. Такий підхід дозволяє гарантувати надійність і якість системи в цілому [5,6].

3.2 Інструменти для автоматизованого тестування

Для автоматизації тестування системи Linguist Portal було обрано два інструменти, які є визнаними лідерами у сфері забезпечення якості програмного забезпечення та зарекомендували себе як ефективні рішення для перевірки як веб-інтерфейсів, так і серверних API. Це Cypress, який використовується для тестування клієнтської частини системи (веб-інтерфейсу), та Postman, призначений для перевірки функціональності прикладного програмного інтерфейсу (API) [5,6]. Ці інструменти стали стандартами в галузі завдяки зручності використання, розширеному функціоналу та підтримці сучасних методологій розробки. Їхнє застосування в проєкті Linguist Portal забезпечує комплексне тестування взаємодії користувачів із системою через браузер і технічної коректності серверної логіки через API-запити.

Cypress — це передова платформа для автоматизації тестування, яка підтримує широкий спектр перевірок, включаючи наскрізні (end-to-end) тести, що відтворюють дії реального користувача від початку до кінця, інтеграційні тести для перевірки взаємодії компонентів, а також модульні тести для ізольованої перевірки окремих функцій. Ключовою особливістю Cypress є виконання тестів у реальному браузері, що дозволяє точно симулювати дії користувача, такі як кліки, введення даних у форми, перехід між сторінками чи очікування завантаження елементів. Це забезпечує високу достовірність результатів і допомагає виявляти

проблеми, пов’язані з конкретними браузерами. Порівняно з іншими інструментами, Cypress вирізняється інтуїтивно зрозумілим синтаксисом, ефективними засобами для налагодження, можливістю створювати власні команди, вичерпною документацією та підтримкою активної спільноти. Крім того, інструмент інтегрується з системами безперервної інтеграції (CI) та безперервного розгортання (CD), що дозволяє автоматизувати тестування в рамках життєвого циклу розробки. У проєкті Linguist Portal Cypress застосовується для перевірки основних сценаріїв користувача, таких як авторизація, навігація між модулями системи, створення задач, заповнення форм, завантаження файлів і контроль відображення даних.

Postman, у свою чергу, є універсальним інструментом для тестування API, який підтримує створення, виконання та автоматизацію HTTP-запитів до RESTful ендпоінтів. Цей інструмент дозволяє формувати запити всіх типів (GET, POST, PUT, DELETE тощо), додавати параметри, заголовки, тіло запиту, авторизацію та використовувати змінні для адаптації до різних середовищ. Завдяки вбудованій підтримці JavaScript, Postman дає змогу створювати складні сценарії перевірки, наприклад, аналізувати структуру відповідей, перевіряти статус-коди чи контролювати час виконання запитів. У Linguist Portal Postman використовується для тестування повного циклу роботи із задачами: створення, оновлення, отримання та видалення записів із бази даних із подальшою перевіркою коректності операцій. Інтеграція з утилітою Newman дозволяє запускати тести через командний рядок, що забезпечує їх виконання в автоматизованих CI/CD-процесах після оновлення коду чи розгортання нової версії системи.

Вибір Cypress і Postman для тестування Linguist Portal ґрунтується на їхніх численних перевагах. По-перше, обидва інструменти відповідають сучасним підходам до розробки, таким як DevOps, Agile та CI/CD. По-друге, вони мають активну спільноту, що забезпечує доступ до численних прикладів, плагінів і рішень типових проблем. По-третє, завдяки простоті інтерфейсу та якісній документації, ці інструменти доступні навіть для фахівців із початковим досвідом тестування, що полегшує їхнє впровадження. Найважливіше, що використання Cypress і

Postman дозволяє досягти високого рівня якості, стабільності та функціональності системи Linguist Portal, забезпечуючи надійну роботу з лінгвістичними задачами, коректність бізнес-логіки та комфортний досвід для користувачів. Таким чином, ці інструменти є ключовими елементами стратегії автоматизації тестування в проєкті.

3.3 Тестування web інтерфейсу

Тестування web інтерфейсу Linguist Portal за допомогою Cypress має на меті забезпечити стабільність, функціональність і зручність використання користувацького інтерфейсу. Цей процес охоплює перевірку всіх ключових елементів, які впливають на взаємодію користувача з системою. Тестування зосереджено на перевірці коректності входу в систему з правильними та неправильними обліковими даними, обробки помилок, роботи форм для введення даних, їхньої валідації та збереження, а також правильності переходів між сторінками, функціонування меню, фільтрів і відображення звітів, таблиць і графіків [5].

Для реалізації автоматизованого тестування web інтерфейсу, яке включало запуск додатку та процес авторизації, було задіяно інструменти Cypress, Node.js та Visual Studio Code. Платформа Node.js забезпечувала виконання JavaScript-коду, що є ключовою умовою для стабільної роботи фреймворку Cypress. Редактор Visual Studio Code слугував для розробки та модифікації тестових файлів, значно спрощуючи створення автоматизованих сценаріїв тестування.

На першому етапі було перевірено правильність встановлення Node.js за допомогою команди `npm -v`. Встановлення представлено на рисунку 3.1.

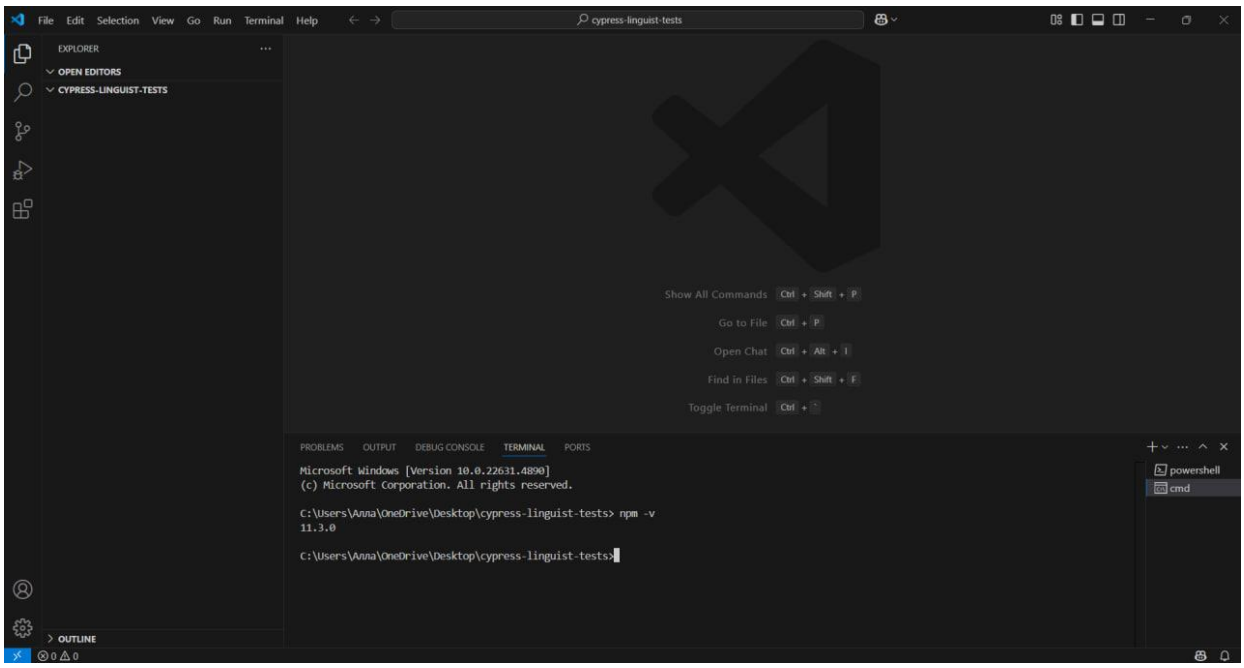

Рисунок 3.1— інтерфейс Visual Studio Code при застосуванні команди `npm -v` в термінал

Далі, у спеціально підготовленій директорії проєкту, створено файл `package.json` командою `npm init -y`, що дало змогу визначити базову конфігурацію проєкту. Створення файлу `package.json` зображено на рисунку 3.2.

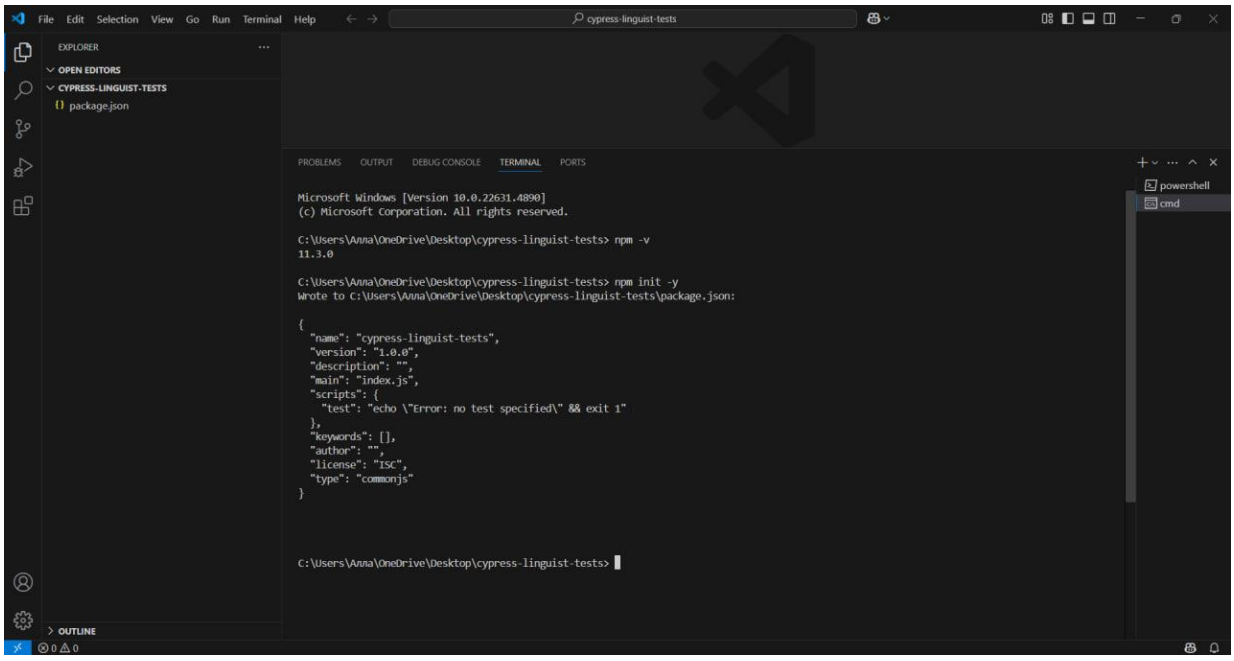

Рисунок 3.2— інтерфейс Visual Studio Code при застосуванні команди `npm init -y` у термінал

Після цього проведено інсталяцію Cypress за допомогою команди `npm install cypress --save-dev`, додавши фреймворк до переліку залежностей проєкту. Інсталяцію Cypress в редакторі Visual Studio Code зображена на рисунку 3.3.

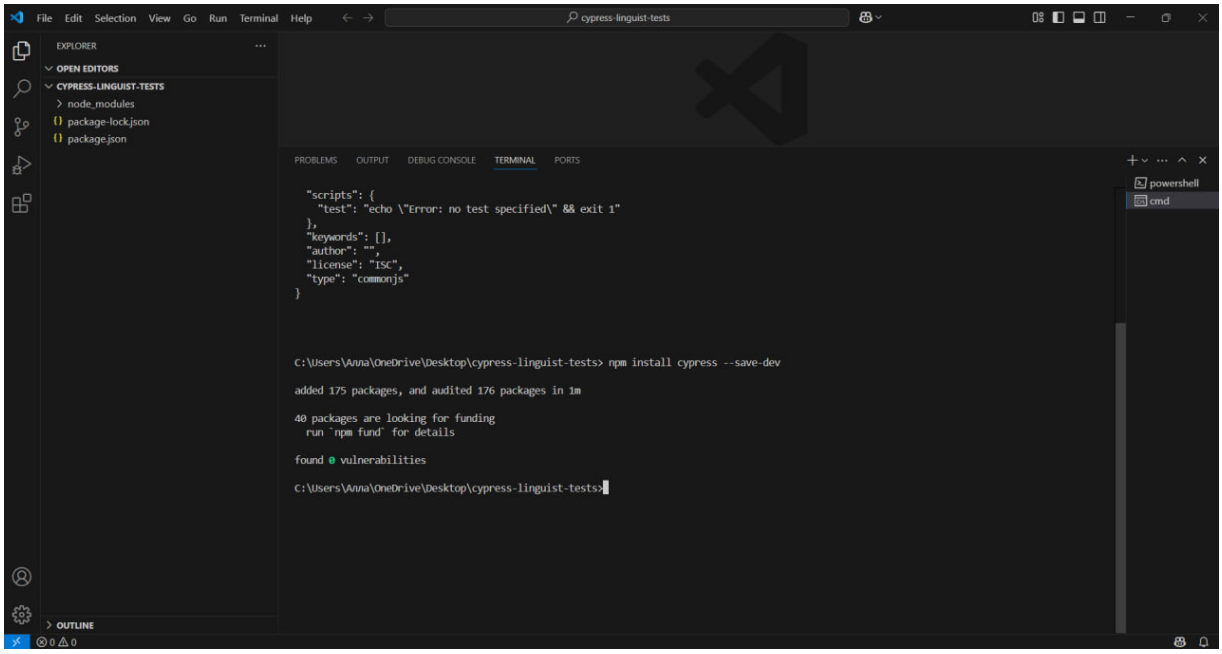

Рисунок 3.3— інтерфейс Visual Studio Code при інсталяції Cypress за допомогою команди через термінал

У межах сформованої структури Cypress створено підпапку `e2e`, де розміщено файл `login.cy.js`. У цьому файлі за допомогою JavaScript реалізовано автоматизований сценарій для тестування функції авторизації. Тестування охопило два основні сценарії. Перший, позитивний, перевіряв можливість входу в систему при введенні коректних облікових даних, з подальшою верифікацією переходу на головну сторінку після успішного входу. Результати виконання коду та успішного тестування зображено на рисунку 3.4 та рисунку 3.5.

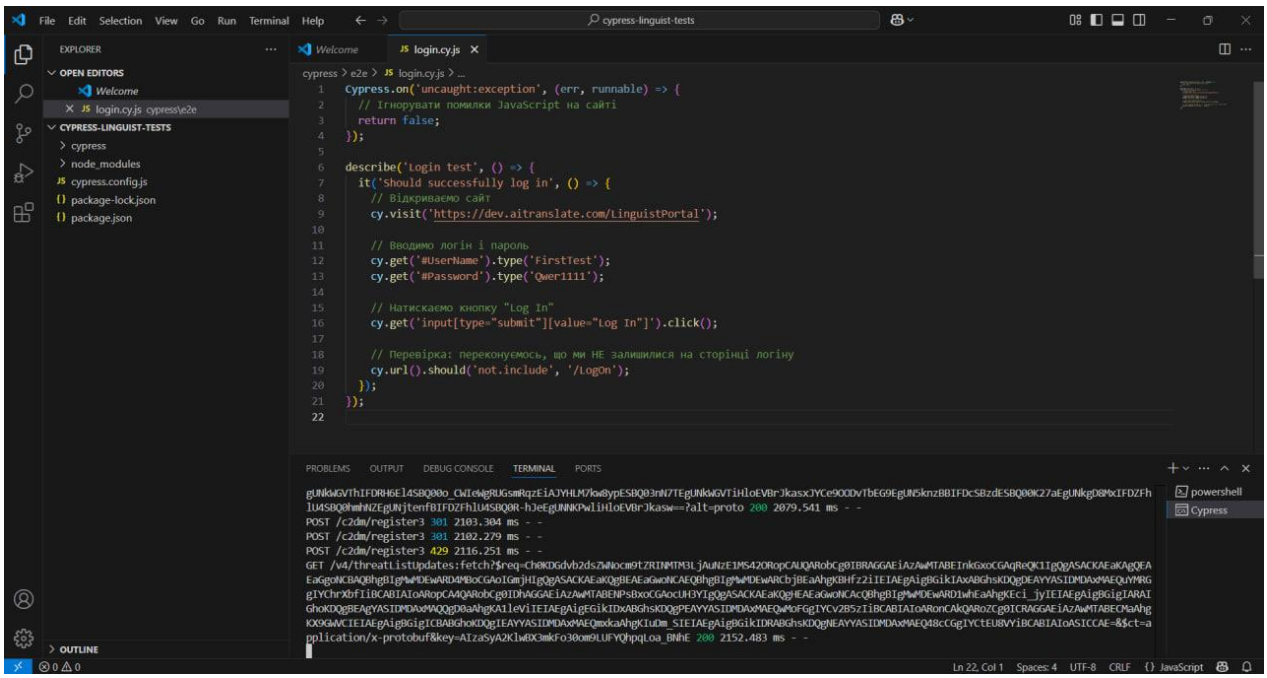

Рисунок 3.4 — виконання коду у Visual Studio Code для створення позитивного тест-кейсу

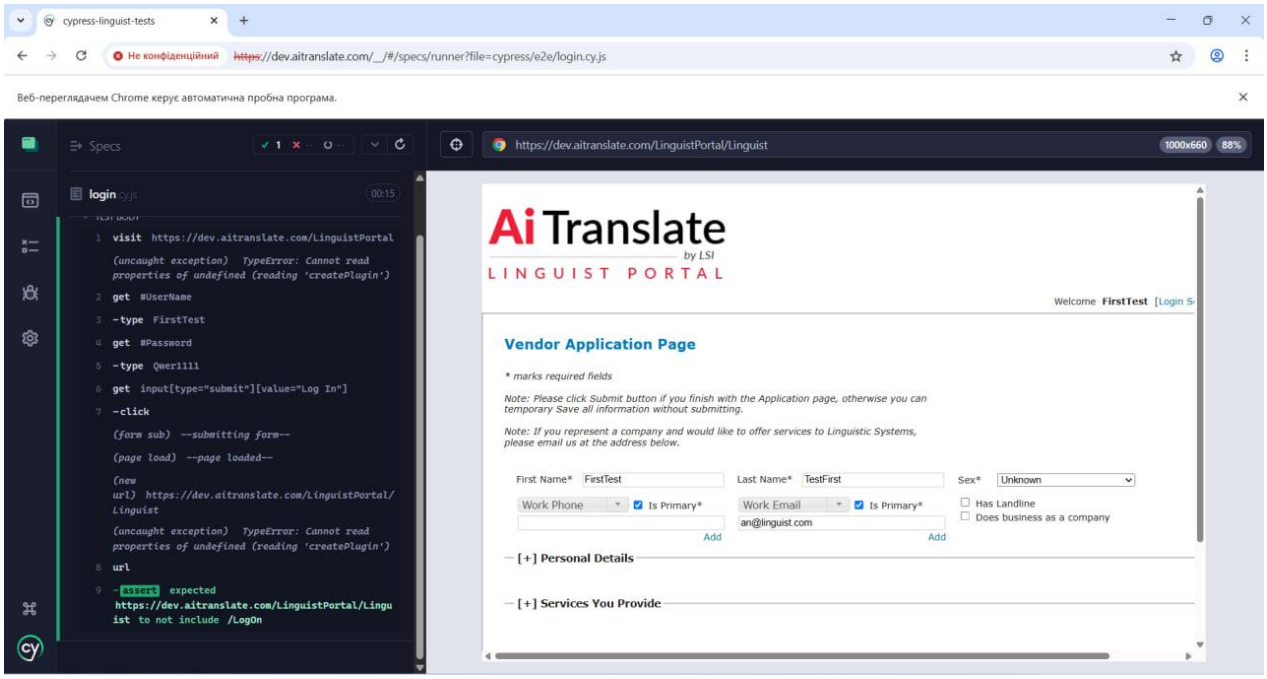

Рисунок 3.5 — успішний результат тестування на вхід в систему з валідними даними

Другий, негативний, тестував поведінку системи при використанні неправильного логіна чи пароля, аналізуючи появу повідомлення про помилку та бло-

кування доступу до внутрішнього інтерфейсу. Створений код для тестування негативного тест-кейсу та результати його тестування представлені на рисунку 3.6 та рисунку 3.7.

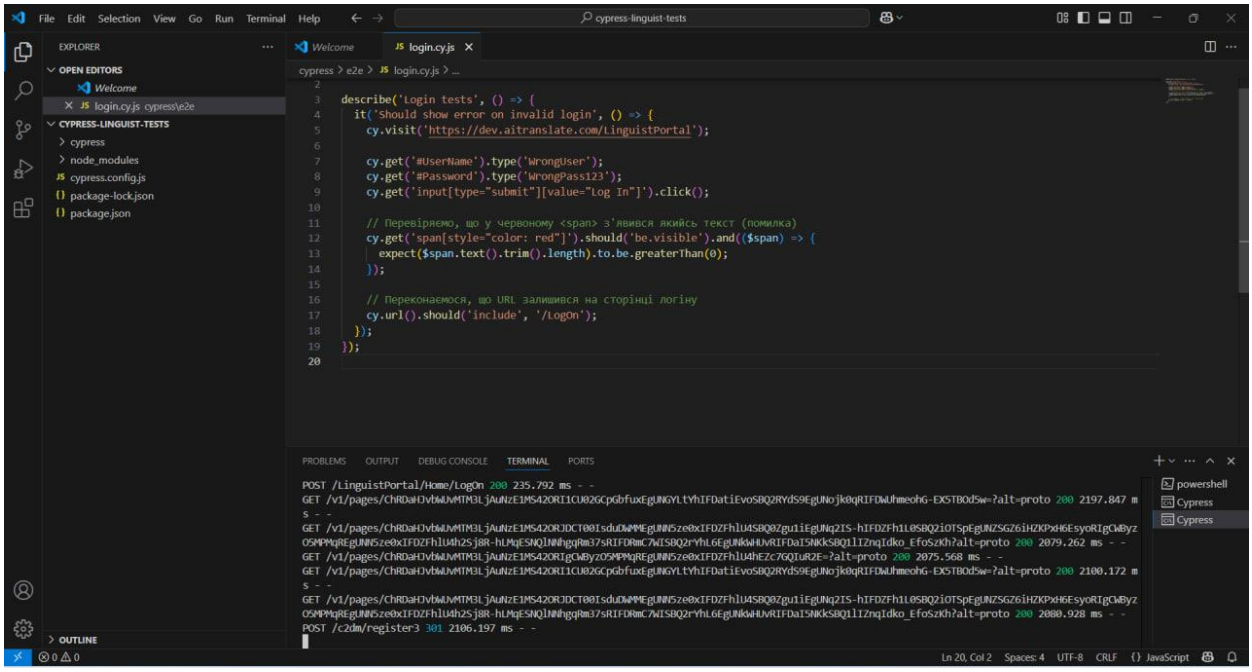

Рисунок 3.6 — виконання коду у Visual Studio Code для створення негативного тест-кейсу

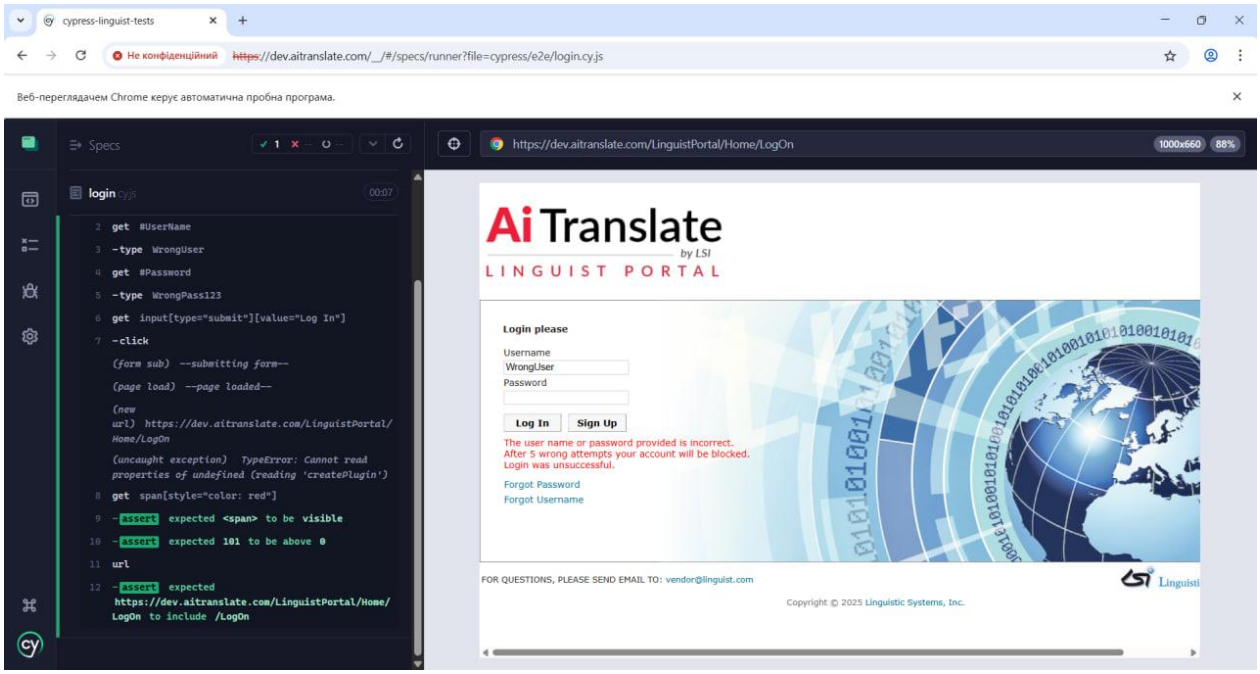

Рисунок 3.7— успішний результат тестування вебдодатку при вході в систему з невалідними даними

Отже, проведені кроки забезпечили виконання базового функціонального тестування процесу авторизації вебдодатку, що є важливим етапом взаємодії користувача з системою.

3.4 Тестування API

Тестування API Linguist Portal за допомогою Postman спрямоване на перевірку коректності роботи серверної частини системи, яка є основою для взаємодії між web інтерфейсом і базою даних, а також для інтеграції з іншими системами. Тестування охоплює перевірку запитів GET, POST, PUT і DELETE для створення, отримання, оновлення та видалення лінгвістичних задач, а також валідацію статус-кодів (200, 201, 400, 404 тощо), формату даних (JSON) і їхньої відповідності специфікаціям. Окрема увага приділяється перевірці поведінки API при некоректних вхідних даних або відсутності авторизації [6].

Для перевірки роботи вебпорталу Linguist Portal було використано програму Postman, яка дозволяє надсилати HTTP-запити до сервера. Було здійснено GET-запит за адресою https://dev.aitranslate.com/LinguistPortal/?ReturnUrl=%2fLinguistPortal%2fLinguist, який викликав завантаження HTML-коду сторінки. Сервер повернув код успішної відповіді 200 OK, що свідчить про правильну роботу ресурсу. Також був переданий параметр ReturnUrl, який використовується для переадресації після входу користувача. Тестування серверу при завантажені сторінки вебдодатку зображено на рисунку 3.8.

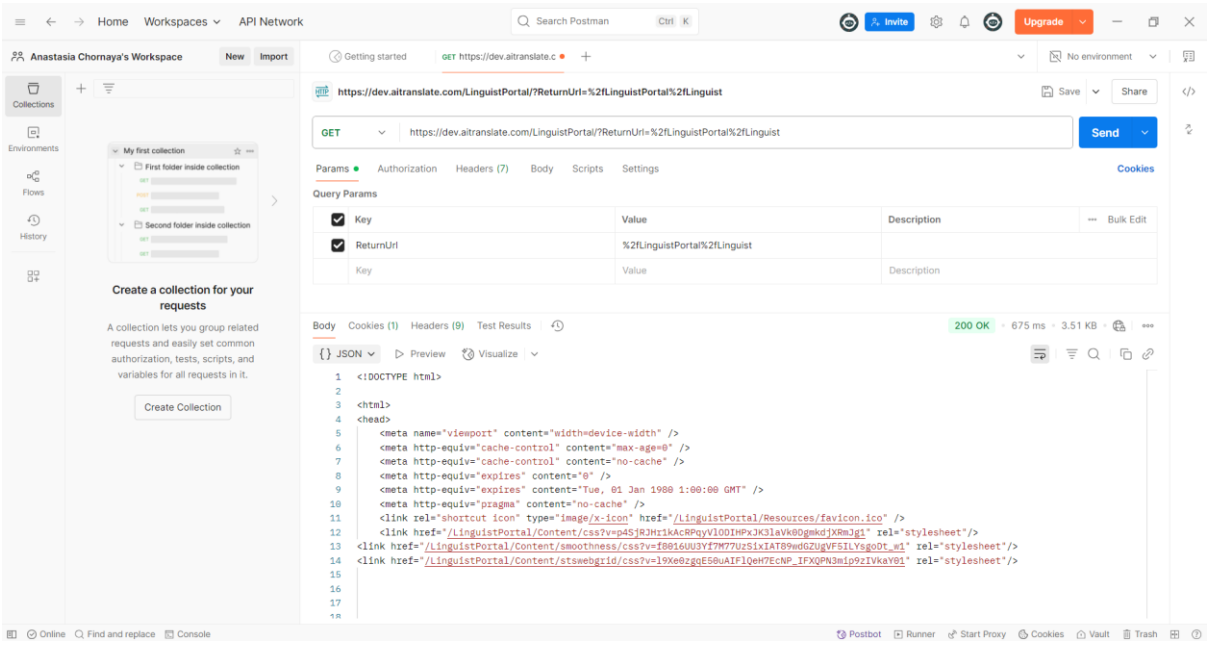

Рисунок 3.8— перевірка роботи серверу для входу в систему

Для створення позитивного тест-кейсу було здійснено POST-запит з метою перевірки автентифікації користувача. У запиті було використано метод POST і передано такі параметри: Username: “FirstTest”, Password: “Qwer1111”, ReturnUrl: “/LinguistPostal”. У результаті сервер повернув статус 200 OK і HTML-код наступної сторінки, що підтверджує правильність авторизації. Це свідчить про те, що система успішно обробляє вхідні дані для входу в систему. Тестування серверу при введені коректних даних представлено на рисунку 3.9.

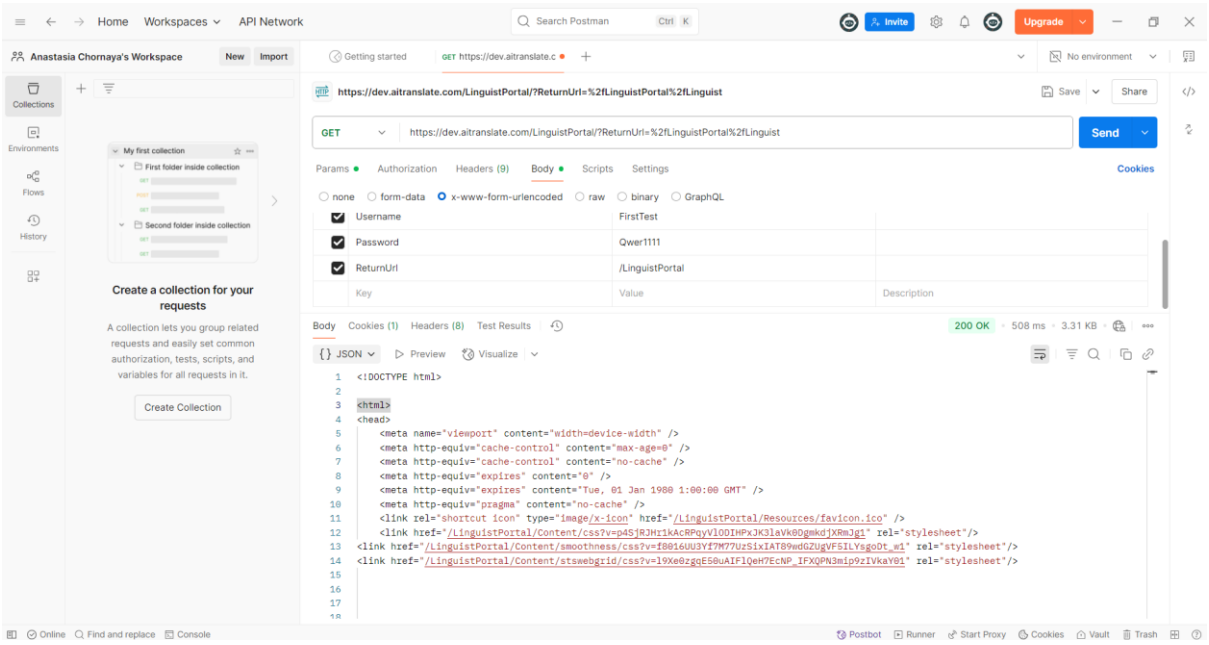

Рисунок 3.9— успішне тестування серверу при введені валідних даних

Для створення негативного тест-кейсу було проведено POST-запит з метою знаходження помилки при вході в систему з некоректними даними. У разі неправильного введення логіна або пароля, сервер повертає статус 200 OK, а в тілі HTML відображає повідомлення про помилку: ., що свідчить про коректну роботу валідації на стороні сервера. Тестування роботи серверу при введені некоректних даних зображено на рисунку 3.10.

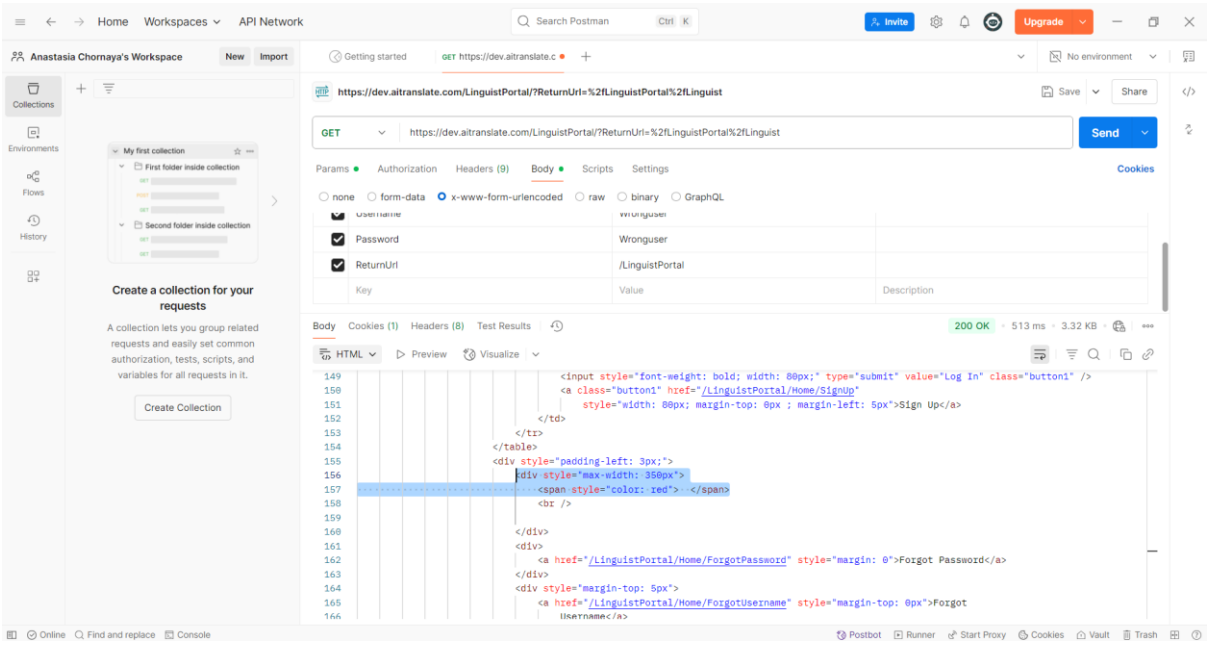

Рисунок 3.10— Тестування серверу при вході в систему з невалідними даними

Під час тестування API системи Linguist Portal було перевірено коректність обробки запитів на авторизацію. Було виявлено, що сервер повертає код відповіді 200 навіть у випадку помилкових даних, що свідчить про правильну технічну роботу ендпоінту. Однак повідомлення про помилку передається у тілі HTML-сторінки, а не у вигляді статус-коду, що потребує додаткової перевірки вмісту відповіді. Такий підхід дозволив оцінити поведінку системи при різних сценаріях і підтвердив важливість тестування API як ключового етапу перевірки надійності вебдодатків.

ВИСНОВКИ

У процесі виконання бакалаврської роботи було проведено детальне дослідження методів тестування вебдодатків, з особливим акцентом на автоматизований підхід до оцінки функціональності, продуктивності та безпеки сучасних вебсистем. У першому розділі висвітлено основи тестування вебдодатків, підкреслено його важливість для забезпечення якості програмного забезпечення, проаналізовано наслідки ігнорування тестування, а також наведено класифікацію методів і життєвий цикл цього процесу.

Другий розділ зосереджено на автоматизованому тестуванні. У ньому розглянуто ключові принципи, різновиди, методи та типи автоматизації тестування. Окрема увага приділена сучасним інструментам, зокрема Selenium, Cypress, Playwright, Postman, Apache JMeter, Locust, k6, OWASP ZAP, а також засобам інтеграції тестування в процес CI/CD — Jenkins і Docker. Аналіз цих інструментів став підставою для обґрунтування вибору найбільш придатних для практичного використання в проєкті.

У третьому розділі здійснено практичне дослідження — автоматизоване тестування web інтерфейсу та API системи управління лінгвістичними задачами Linguist Portal. Описано архітектуру додатку, визначено об’єкти тестування та обґрунтовано вибір відповідних інструментів. Для перевірки функціональності інтерфейсу застосовано метод енд-ту-енд (end-to-end) тестування, що забезпечило оцінку повного сценарію взаємодії користувача з системою — як у позитивних, так і в негативних випадках. Окрім того, проведено тестування API, зокрема перевірено реакцію сервера на запити авторизації з правильними та неправильними даними. Вибрані тест-кейси відображають основні можливості системи, зокрема процес автентифікації, і забезпечують оцінку як зовнішньої взаємодії інтерфейсу, так і реакції сервера. Це дає змогу досягти початкового рівня довіри до коректної роботи модуля автентифікації.

Загалом, виконана робота підтвердила доцільність використання автоматизованих методів тестування вебдодатків, що сприяє не лише підвищенню якості програмного продукту, а й суттєвому вдосконаленню процесу його перевірки. Автоматизація дозволяє скоротити час тестування функціоналу щонайменше в 3–5 разів порівняно з ручним методом, а при повторних тестах — більш ніж у 10 разів. Цей підхід знижує ризик людських помилок, гарантує стабільність результатів і сприяє ранньому виявленню критичних дефектів до випуску продукту. У контексті великих або часто оновлюваних систем автоматизація забезпечує значну економію ресурсів і є невід’ємною частиною ефективного процесу розробки та підтримки вебдодатків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 The Automated Testing Handbook, Software Test Professionals (Soft-TestPro), 2009 https://www.softwaretestpro.com/wp-content/uploads/2016/10/AutomatedTestingHandbook.pdf. – (дата звернення 14.04.2025)

2 Galin D. Software Quality Assurance: From Theory to Implementation / D. Galin. – Harlow : Pearson Education, 2018 https://hero.lecturer.pens.ac.id/datahero/kuliah/MKPL/Software%20Quality%20Assurance%20From%20Theory%20to%20Implementation.pdf. – (дата звернення 12.03.2025).

3 Iva Kertusha, Gebremariem Assress, Onur Duman, Andrea Arcuri. “A Survey on Web Testing: On the Rise of AI and Applications in Industry” https://www.ijert.org/research/a-comprehensive-review-on-selenium-automation-testing-tool-IJERTCONV5IS20027.pdf. – (дата звернення 13.03.2025).

4 Selenium Project Documentation https://www.selenium.dev/documentation. – (дата звернення: 13.03.2025).

5 Cypress Documentation https://docs.cypress.io/. – (дата звернення: 13.03.2025).

6 Playwright Documentation https://playwright.dev/. – (дата звернення: 13.03.2025).

7 Apache JMeter User Manual https://jmeter.apache.org/usermanual/. – (дата звернення: 14.03.2025).

8 Locust Documentation https://docs.locust.io/. – (дата звернення: 14.03.2025).

9 k6 Performance Testing https://k6.io/docs/. – (дата звернення: 15.03.2025).

10 OWASP ZAP Project https://owasp.org/www-project-zap/. – (дата звернення: 16.03.2025).

11 Jenkins User Documentation https://www.jenkins.io/doc/. – (дата звернення: 17.03.2025).

12 Docker Documentation https://docs.docker.com/. – (дата звернення: 17.03.2025).

13 ISTQB Glossary https://glossary.istqb.org/. – (дата звернення: 22.03.2025).

14 dev.aitranslate.com – https://dev.aitranslate.com/LinguistPortal. – (дата звернення: 10.05.2025).

15 "9 Test Automation Benefits – Tricentis" https://www.tricentis.com/learn/top-9-test-automation-benefits . – (дата звернення: 14.04.2025).

16 "Playwright vs Selenium vs Cypress: A Detailed Comparison" https://www.lambdatest.com/blog/playwright-vs-selenium-vs-cypress/ (дата звернення: 16.04.2025).

17 "Postman vs. Other API Testing Tools: A Comparative Analysis" https://www.geeksforgeeks.org/postman-vs-other-api-testing-tools-a-comparative-analysis/. (дата звернення: 18.04.2025).

18 Чорна А. О., Богач І. В. Застосування технології автоматизованого тестування для системи управління лінгвістичними задачами Linguist Portal [Матеріали конференції] // LIV Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації. Вінниця, 2025. https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/25607 (дата звернення: 14.06.2025).

Додатки

Додаток А

(обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

ЗАСТОСУВАННЯ ТЕХНОЛОГІЇ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ДЛЯ СИСТЕМИ УПРАВЛІННЯ ЛІНГВІСТИЧНИМИ ЗАДАЧАМИ LINGUIST PORTAL

Варіанти використання Linguist Portal

Рисунок А.1 – UML-діаграма варіантів використання Linguist Portal

UML-діаграма класів Linguist Portal

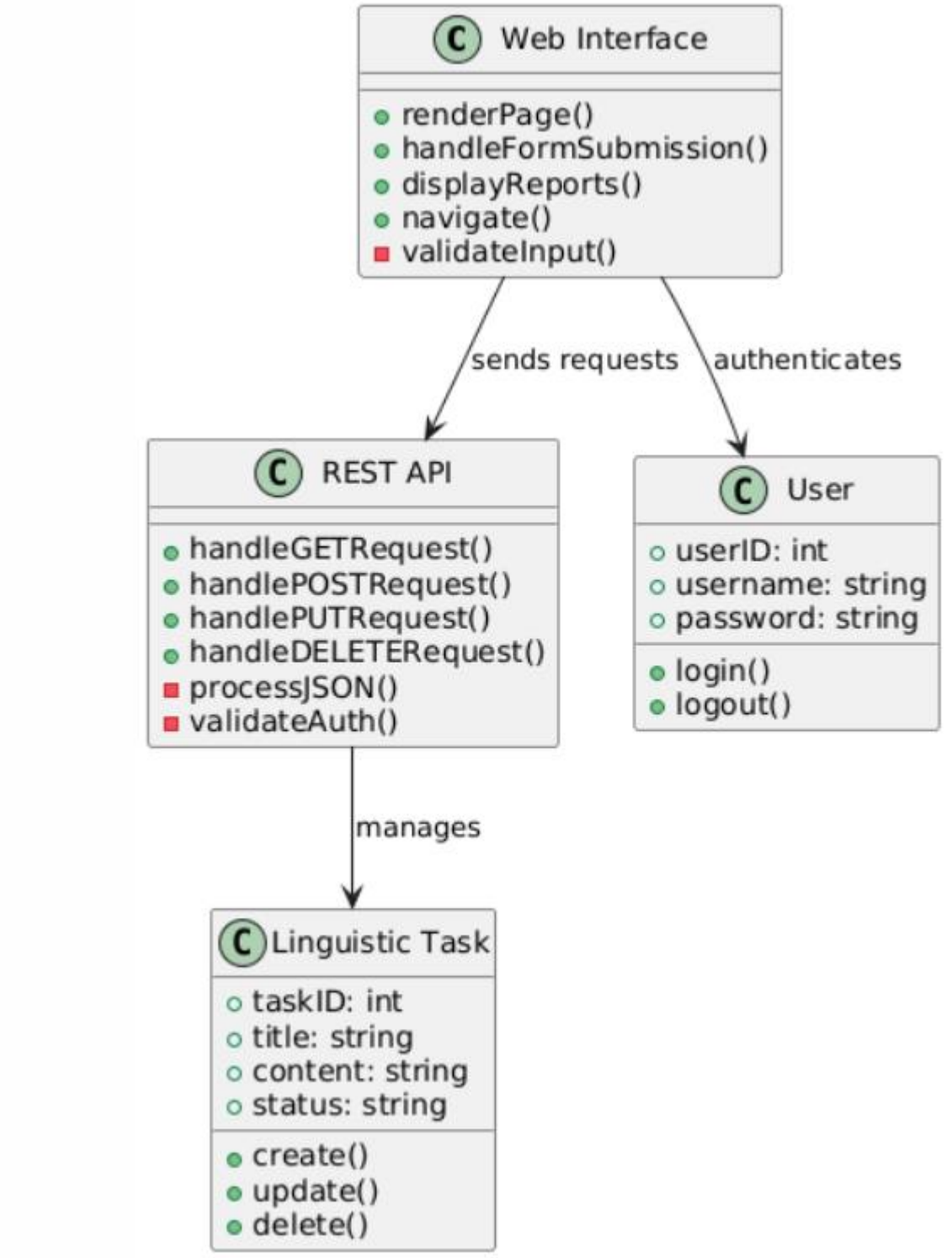

Рисунок А.2 - UML-діаграма класів Linguist Portal

UML-діаграма активності автоматизованого тестування Linguist Portal

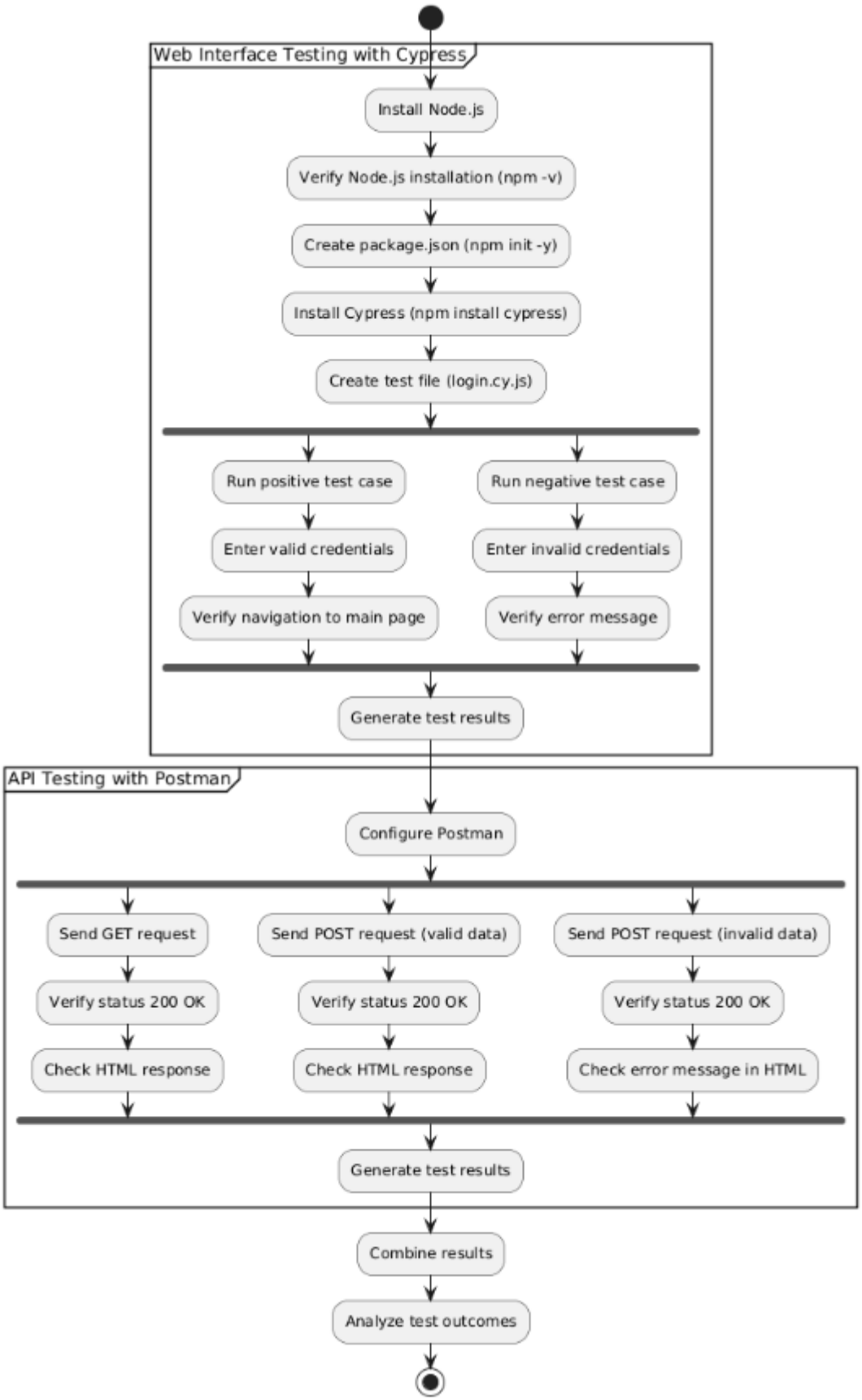

Рисунок А.3 – UML-діаграма активності автоматизованого тестування Linguist Portal

Додаток Б

(обов’язковий)

Лістинг тестування

```
Cypress.on('uncaught:exception', () => false);


describe('Login tests', () => {
  it('Should show error on invalid login', () => {
    cy.visit('https://dev.aitranslate.com/LinguistPortal');


    cy.get('#UserName').type('WrongUser');
    cy.get('#Password').type('WrongPass123');
    cy.get('input[type="submit"][value="Log In"]').click();


    // Перевіряємо, що у червоному <span> з'явився якийсь текст (помилка)
    cy.get('span[style="color: red"]').should('be.visible').and(($span) => {
      expect($span.text().trim().length).to.be.greaterThan(0);
    });


    // Переконаємося, що URL залишився на сторінці логіну
    cy.url().should('include', '/LogOn');
  });
});
```

63

Додаток В

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Застосування технології автоматизованого тестування для системи управління лінгвістичними задачами Linguist Portal

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ АІІТ, ФІІТА, 1АКІТ-21Б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 0,00 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

☐ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Олег БІСІКАЛО, зав. кафедри АІІТ _____ (підпис)
(прізвище, ініціали, посада)

Любов ЯНЧУК, секретар кафедри АІІТ _____ (підпис)
(прізвище, ініціали, посада)

Особа, відповідальна за перевірку _____ Костянтин ОВЧИННИКОВ
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Роман КВЄТНИЙ, професор кафедри АІІТ
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Анастасія ЧОРНА
(підпис) (прізвище, ініціали)

Науково-виробниче підприємство
“СПІЛЬНА СПРАВА” Товариство з обмеженою відповідальністю
Україна, 21000, м. Вінниця, вул. Магістратська, 36/3, тел. +38(096)-558-24-64
код ЄДРПОУ 31041670, код платників податків 310416702282, свідоцтво № 0183329
IBAN : UA 41 322313 0000026004000003226 в філії АТ «Укрексімбанк» в м. Вінниці

Затверджую

Директор

НВП «СПІЛЬНА СПРАВА», ТОВ

Малачковська Роксолана Ігорівна

«05» червня 2025 р.

АКТ

впровадження результатів бакалаврської роботи

Чорної Анастасії Олегівни «Застосування технології автоматизованого тестування для системи управління лінгвістичними задачами Linguist Portal»

Комісія у складі головного спеціаліста, керівника відділу обробки даних С. Житанського та керівника відділу розробки інформаційних систем, О. Кириленка склали цей акт про те, що у Науково-виробничому підприємстві “Спільна Справа”, ТОВ впроваджуються результати, які отримані бакалавром Чорною А.О. при виконанні бакалаврської кваліфікаційної роботи мають практичну цінність. Технології автоматизованого тестування сприяють підвищенню ефективності процесу тестування, прискорюють процес та дають можливість уникати помилки ручного тестування.

Таким чином, актуальність роботи А.О. Чорної не викликає сумнівів, а низка методик, підходів та результати їх застосування можуть бути використаними у практичній діяльності.

Науково-виробничому підприємству “Спільна Справа”, ТОВ бакалавром Чорною А. О. передано програмне забезпечення, яке дозволяє підвищити ефективність процесу автоматизованого тестування.

Члени комісії:

Головний спеціаліст Сергій ЖИТАНСЬКИЙ

Керівник відділу Олександр КИРИЛЕНКО