Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
бакалавр
(ступінь вищої освіти)

на тему СТВОРЕННЯ ПРОГРАМНОЇ СИСТЕМИ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ВЕБЗАСТОСУНКІВ НА ОСНОВІ АНАЛІЗУ HTTP-ЗАПИТІВ
DESIGN OF AN AUTOMATED TESTING SYSTEM FOR WEB APPLICATIONS BASED ON HTTP REQUEST ANALYSIS

Виконав(ла): студент(ка) 4 курсу, групи КНТ-222
Спеціальності 122 Комп'ютерні науки
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Комп'ютерні науки

ЦВЄТКОВ Т.О.
(ПРІЗВИЩЕ та ініціали)
Керівник ГЛАДКОВА О.М.
(ПРІЗВИЩЕ та ініціали)
Рецензент ГОЛУБ Т.В.
(ПРІЗВИЩЕ та ініціали)

2026

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
_____Сергій СУББОТІН
"____" _____2026 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ЦВЄТКОВА Тимура Олеговича
(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Створення програмної системи автоматизованого тестування вебзастосунків на основі аналізу HTTP-запитів. Design of an Automated Testing System for Web Applications Based on HTTP Request Analysis

керівник проєкту (роботи) к.т.н., доцент, ГЛАДКОВА Ольга Миколаївна,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)
затверджені наказом закладу вищої освіти від " 07 " квітня 2026 року № 139
2. Строк подання студентом проєкту (роботи) 10 червня 2026 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області та методів тестування вебзастосунків. 2. Методи і засоби аналізу HTTP-запитів. 3. Розробка програмної системи тестування. 4. Експериментальне дослідження та аналіз результатів.

5. Перелік графічного матеріалу (з з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ГЛАДКОВА О.М., доцент		
Нормоконтроль	ДЕЙНЕГА Л.Ю., ст. викладач		

7. Дата видачі завдання « 20 » квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій	2 тиждень	Розділ 2
	розробки.		
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження	5 тиждень	Розділ 4
	програмного забезпечення.		
7	Оформлення пояснювальної записки та документів	6 тиждень	Додатки
	до неї.		
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка) _____ Тимур ЦВЄТКОВ
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ Ольга ГЛАДКОВА
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 64 с., 6 табл., 21 рис., 2 дод., 12 джерел.

ТЕСТУВАННЯ, HTTP-ЗАПИТИ, REST API, POSTMAN, NEWMAN, ВЕБЗАСТОСУНОК, JSON, API-ТЕСТУВАННЯ.

Об’єкт дослідження – методи та програмні системи аналізу HTTP-запитів у вебзастосунках для тестування REST API.

Предмет дослідження – методи та програмні засоби тестування вебзастосунків на основі аналізу HTTP-запитів.

Мета роботи – зменшення часу виявлення помилок у REST API вебзастосунків шляхом створення програмної системи тестування на основі аналізу HTTP-запитів.

Матеріали, методи та технічні засоби: HTTP-протокол, REST API, середовище Postman, інструмент Newman, Chrome DevTools, JSON, JavaScript, операційна система Windows.

Результати. Розроблено програмну систему тестування REST API вебзастосунків, яка дозволяє аналізувати HTTP-запити, виконувати перевірки відповідей сервера та формувати звіти про результати тестування. Реалізовано набір API-тестів для перевірки CRUD-операцій, авторизації, статус-кодів та структури JSON-відповідей.

Висновки. Розроблена система тестування дозволяє скоротити час перевірки REST API, зменшити кількість помилок при ручному тестуванні та підвищити стабільність вебзастосунків.

Галузь використання –тестування REST API вебзастосунків у командах розробки та забезпечення якості програмного забезпечення.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 64 pages, 6 tables, 21 figures, 2 appendixes, 12 sources.

TESTING, HTTP REQUESTS, REST API, POSTMAN, NEWMAN, WEB APPLICATION, JSON, API TESTING.

Object of study is methods and software systems for HTTP request analysis in web applications for REST API test.

Subject of study is methods and software tools for testing of web applications based on HTTP request analysis.

The purpose of the work is to reduce the time required to detect errors in REST APIs of web applications by creating a software system for testing based on HTTP request analysis.

Materials, methods, and tools: HTTP protocol, REST API, Postman environment, Newman tool, Chrome DevTools, JSON, JavaScript, Windows operating system.

Results. A software system for REST API testing of web applications was developed. The system allows analyzing HTTP requests, performing automated validation of server responses, and generating testing reports. A set of API tests for CRUD operations, authorization, HTTP status codes, and JSON response structure validation was implemented.

Conclusions. The developed testing system reduces the time required for REST API verification, decreases the number of manual testing errors, and improves the stability of web applications.

The field of use is REST API testing of web applications in software development and quality assurance teams.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface;

CRUD – створення (Create), читання (Read), оновлення (Update) та видалення (Delete);

CI/CD – Continuous Integration / Continuous Delivery;

GUI – Graphical User Interface;

HTTP – HyperText Transfer Protocol;

IDE – Integrated Development Environment,

JSON – JavaScript Object Notation;

JWT – JSON Web Token;

REST – Representational State Transfer;

URL – Uniform Resource Locator;

XHR – XML HTTP Request;

ПЗ – програмне забезпечення.

ВСТУП

У сучасних умовах розвитку інформаційних технологій вебзастосунки стали основою більшості цифрових сервісів. Значна частина сучасного програмного забезпечення використовує REST API для обміну даними між клієнтською та серверною частинами системи [1]. Зі збільшенням складності вебзастосунків та кількості HTTP-запитів виникає потреба у швидкому та надійному тестуванні програмних інтерфейсів, що забезпечують взаємодію між компонентами системи.

Традиційне ручне тестування REST API потребує значних витрат часу та ресурсів, особливо при регулярних оновленнях програмного забезпечення. Крім того, ручне тестування підвищує ризик виникнення людських помилок і ускладнює проведення регресійного тестування. У зв’язку з цим автоматизоване тестування REST API є одним із ключових напрямів забезпечення якості сучасного програмного забезпечення [2].

Особливу роль у процесі тестування відіграє аналіз HTTP-запитів, який дозволяє визначати структуру взаємодії між клієнтом і сервером, виявляти параметри запитів, заголовки, методи авторизації та формати передачі даних. Аналіз HTTP-трафіку через інструменти браузера, зокрема Chrome DevTools, дозволяє отримувати інформацію про роботу REST API навіть за відсутності технічної документації [3].

Для автоматизації тестування REST API широко використовуються сучасні програмні засоби, серед яких одним із найпопулярніших є Postman. Даний інструмент дозволяє створювати HTTP-запити, реалізовувати автоматизовані сценарії перевірок за допомогою JavaScript та формувати колекції тестів [2]. Для автоматичного запуску тестів та генерації звітності використовується Newman, що забезпечує інтеграцію API-тестування у процеси безперервної інтеграції та доставки програмного забезпечення.

Актуальність теми зумовлена необхідністю скорочення часу тестування REST API вебзастосунків, зменшення кількості помилок при

ручному тестуванні та підвищення стабільності сучасних вебсистем шляхом автоматизації процесів перевірки HTTP-запитів і відповідей сервера.

Об’єкт дослідження – методи та програмні системи аналізу HTTP-запитів у вебзастосунках для автоматизації тестування REST API.

Предмет дослідження – методи та програмні засоби автоматизованого тестування вебзастосунків на основі аналізу HTTP-запитів.

Мета роботи – зменшення часу виявлення помилок у REST API вебзастосунків шляхом створення програмної системи автоматизованого тестування на основі аналізу HTTP-запитів.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– проаналізувати особливості HTTP-протоколу та REST API вебзастосунків;

– дослідити сучасні методи автоматизованого тестування вебзастосунків;

– обґрунтувати вибір програмних засобів для реалізації системи тестування;

– розробити структуру програмної системи автоматизованого тестування;

– реалізувати автоматизовані сценарії тестування HTTP-запитів;

– провести експериментальне дослідження роботи системи та оцінити скорочення часу тестування.

Практичне значення роботи полягає у створенні програмної системи автоматизованого тестування REST API вебзастосунків, яка дозволяє виконувати автоматизовану перевірку HTTP-запитів, аналізувати відповіді сервера та формувати звіти про результати тестування.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА МЕТОДІВ ТЕСТУВАННЯ ВЕБЗАСТОСУНКІВ

1.1 Особливості HTTP-протоколу та REST API

Сучасні вебзастосунки базуються на постійній взаємодії між клієнтською та серверною частинами системи. Для передачі даних у більшості вебсервісів використовується протокол HTTP (HyperText Transfer Protocol), який є основою функціонування мережі Інтернет. HTTP визначає правила передачі повідомлень між клієнтом і сервером, а також структуру запитів та відповідей [1].

Основною особливістю HTTP є модель «клієнт–сервер», у якій клієнт надсилає запит, а сервер повертає відповідь. У вебзастосунках клієнтом зазвичай виступає браузер або мобільний застосунок, а сервер виконує обробку даних та бізнес-логіки системи. Передача інформації здійснюється за допомогою HTTP-запитів, що містять метод, URL, заголовки та, за необхідності, тіло повідомлення [2].

Структура HTTP-запиту включає метод запиту, адресу ресурсу, HTTP-заголовки, параметри авторизації та тіло запиту.

Найбільш поширеними HTTP-методами є:

– GET – отримання ресурсу;

– POST – створення нового ресурсу;

– PUT – повне оновлення ресурсу;

– PATCH – часткове оновлення ресурсу;

– DELETE – видалення ресурсу [2].

Приклад HTTP-запиту методом GET: GET /api/users HTTP/1.1.

Відповідь сервера зазвичай містить:

– HTTP-статус;

– заголовки;

– тіло відповіді у форматі JSON або XML.

У сучасних вебзастосунках найпоширенішим підходом до побудови серверної архітектури є REST API (Representational State Transfer Application Programming Interface). REST є архітектурним стилем взаємодії компонентів розподілених систем через HTTP-протокол [2].

Основними принципами REST API є:

– ресурсоорієнтованість;
– використання стандартних HTTP-методів;
– відсутність збереження стану між запитами;
– уніфікований інтерфейс взаємодії;
– підтримка кешування [3].

У REST API кожен ресурс має унікальний URI. Наприклад:

– /users;
– /users/15;
– /orders/7.

Такі ендпоінти дозволяють клієнту взаємодіяти з ресурсами системи через стандартизовані HTTP-запити.

Важливою особливістю REST API є використання JSON (JavaScript Object Notation) як основного формату передачі даних. Формат JSON є компактним, зручним для читання та підтримується більшістю сучасних мов програмування [4].

REST API широко використовується у:

– вебзастосунках та мобільних застосунках;
– хмарних сервісах;
– мікросервісній архітектурі;
– системах інтеграції програмного забезпечення.

Популярність REST API обумовлена його простотою, масштабованістю та сумісністю із сучасними вебтехнологіями [4].

У процесі взаємодії клієнта та сервера важливу роль відіграють HTTP-статуси, які повідомляють про результат виконання запиту. Найбільш поширені статус-коди наведено у таблиці 1.1.

Таблиця 1.1 – Основні HTTP-статуси

Статус-код	Назва	Призначення
200	OK	Запит виконано успішно
201	Created	Ресурс успішно створено
400	Bad Request	Некоректний запит
401	Unauthorized	Потрібна авторизація
403	Forbidden	Доступ заборонений
404	Not Found	Ресурс не знайдено
500	Internal Server Error	Внутрішня помилка сервера

Коректне використання HTTP-статусів є важливою складовою проєктування та тестування REST API, оскільки дозволяє клієнтським застосункам правильно обробляти результати запитів [4].

Для аналізу HTTP-запитів сучасні браузери надають вбудовані інструменти розробника, зокрема Chrome DevTools. Вкладка Network дозволяє відстежувати:

– HTTP-запити;
– заголовки;
– статус-коди;
– параметри авторизації;
– час виконання запиту;
– структуру JSON-відповідей.

Такі можливості є важливими для автоматизованого тестування вебзастосунків, оскільки дозволяють досліджувати роботу REST API навіть за відсутності офіційної документації [4].

Таким чином, HTTP-протокол та REST API є фундаментальними технологіями сучасних вебзастосунків. Їх аналіз дозволяє будувати ефективні системи автоматизованого тестування, які забезпечують швидке виявлення помилок, контроль стабільності API та підвищення якості програмного забезпечення.

1.2 CRUD-операції у REST API вебзастосунків

У більшості сучасних вебзастосунків основою роботи з даними є CRUD-операції. CRUD є абревіатурою від Create, Read, Update, Delete – створення, читання, оновлення та видалення даних відповідно. Дані операції є базовими для реалізації взаємодії користувача з ресурсами системи та широко використовуються у REST API [5].

CRUD-модель застосовується у системах керування користувачами, контактних книгах, інтернет-магазинах, банківських сервісах, CRM та ERP системах, соціальних мережах, хмарних сервісах.

У REST API CRUD-операції реалізуються через стандартні HTTP-методи, що дозволяє створити уніфікований механізм взаємодії клієнта та сервера [5].

Відповідність CRUD-операцій HTTP-методам наведено у таблиці 1.2.

Таблиця 1.2 – Відповідність CRUD-операцій HTTP-методам

CRUD-операція	HTTP-метод	Призначення
Create	POST	Створення нового ресурсу
Read	GET	Отримання ресурсу
Update	PUT / PATCH	Оновлення ресурсу
Delete	DELETE	Видалення ресурсу

1.2.1 Операція Create

Операція Create використовується для створення нового ресурсу у системі. У REST API для цього застосовується HTTP-метод POST. Клієнт надсилає на сервер HTTP-запит із тілом повідомлення, що містить дані нового об’єкта [6].

Після успішного створення ресурсу сервер зазвичай повертає:

– статус-код 201 Created;

– ідентифікатор нового ресурсу;

– JSON-відповідь із даними створеного об’єкта.

Операція Create є важливою для тестування, оскільки дозволяє перевіряти:

– коректність валідації даних;

– правильність обробки JSON;

– створення нових записів у системі;

– обробку помилок при некоректних даних.

1.2.2 Операція Read

Операція Read використовується для отримання інформації про ресурси системи. Для цього застосовується HTTP-метод GET [6].

REST API може підтримувати:

– отримання списку ресурсів;

– отримання конкретного ресурсу за ID;

– фільтрацію;

– сортування;

– пагінацію.

Приклад GET-запиту: GET /api/users/2.

Приклад відповіді сервера:

Під час тестування операції Read перевіряються коректність отриманих даних, структура JSON-відповіді, статус-коди, швидкість відповіді сервера, та коректність фільтрації та пагінації.

1.2.3 Операція Update

Операція Update використовується для оновлення існуючого ресурсу. У REST API для цього найчастіше використовуються HTTP-методи PUT або PATCH [6].

Метод PUT застосовується для повного оновлення ресурсу, тоді як PATCH використовується для часткової зміни окремих полів.

Сервер після успішного оновлення повертає:

– статус-код 200 OK;

– оновлений JSON-об’єкт;

– дату оновлення ресурсу.

При тестуванні операції Update перевіряються:

– коректність зміни даних;

– збереження змінених значень;

– обробка неіснуючих ресурсів;

– валідація оновлених полів.

1.2.4 Операція Delete

Операція Delete використовується для видалення ресурсу із системи. Для цього застосовується HTTP-метод DELETE [6].

Приклад DELETE-запиту:

```
DELETE /api/users/2
```

У більшості REST API сервер повертає:

– статус-код 204 No Content;

– порожню відповідь без тіла повідомлення.

Тестування операції Delete дозволяє перевірити коректність видалення ресурсу, обробку повторного видалення, роботу авторизації, та стабільність бази даних після видалення запису.

1.2.5 Особливості тестування CRUD-операцій

CRUD-операції є основою більшості сценаріїв автоматизованого тестування REST API. Саме вони дозволяють перевірити життєвий цикл ресурсу у вебзастосунку [7].

Типовий сценарій тестування CRUD включає:

– створення нового ресурсу;
– отримання створеного ресурсу;
– оновлення даних ресурсу;
– видалення ресурсу;
– перевірку відсутності ресурсу після видалення.

Такі сценарії дозволяють:

– виявляти помилки у бізнес-логіці;
– контролювати стабільність API;
– виконувати регресійне тестування;
– автоматизувати повторювані перевірки.

У сучасних системах автоматизованого тестування CRUD-сценарії реалізуються за допомогою спеціалізованих інструментів, серед яких одними з найпопулярніших є Postman та Newman.

Автоматизація CRUD-тестів дозволяє суттєво скоротити час перевірки REST API та зменшити кількість помилок при ручному тестуванні [7].

Таким чином, CRUD-операції є фундаментальною складовою REST API вебзастосунків. Їх автоматизоване тестування забезпечує контроль правильності роботи сервера, стабільності API та якості програмного забезпечення загалом.

1.3 Методи автоматизованого тестування API

Автоматизоване тестування API є одним із ключових напрямів забезпечення якості сучасного програмного забезпечення. Зі збільшенням кількості вебзастосунків, що використовують REST API для взаємодії між клієнтською та серверною частинами системи, виникає необхідність у швидкій та стабільній перевірці програмних інтерфейсів [7].

На відміну від ручного тестування, автоматизоване тестування дозволяє багаторазово виконувати перевірки без участі користувача, що значно скорочує час тестування та зменшує ризик виникнення людських

помилок. Автотести можуть запускатися після кожної зміни програмного коду, що особливо важливо для сучасних підходів безперервної інтеграції та доставки програмного забезпечення (CI/CD) [7].

API-тестування орієнтоване на перевірку логіки роботи серверної частини застосунку без використання графічного інтерфейсу користувача. Основною метою такого тестування є перевірка:

– коректності HTTP-запитів;
– відповідності статус-кодів;
– структури JSON-відповідей;
– стабільності роботи REST API;
– обробки помилок та виняткових ситуацій.

Автоматизоване тестування API має низку переваг порівняно з тестуванням інтерфейсу користувача:

– вища швидкість виконання тестів;
– менша залежність від змін інтерфейсу;
– можливість раннього тестування серверної логіки;
– простіша інтеграція у CI/CD;
– краща масштабованість тестових сценаріїв [7].

У процесі автоматизованого тестування API застосовуються різні типи тестів, які дозволяють комплексно перевіряти роботу вебзастосунку.

1.3.1 Функціональне тестування API

Функціональне тестування спрямоване на перевірку відповідності роботи API функціональним вимогам системи. Основна увага приділяється правильності виконання CRUD-операцій та коректності відповідей сервера [7].

При функціональному тестуванні перевіряються правильність створення ресурсів, отримання даних, оновлення інформації, видалення записів, та відповідність очікуваної бізнес-логіки.

Приклад функціонального тесту:

– створити нового користувача через POST-запит;

– отримати створеного користувача через GET;

– перевірити відповідність даних.

Функціональне тестування є базовим етапом перевірки REST API та використовується у більшості систем автоматизації тестування.

1.3.2 Негативне тестування

Негативне тестування використовується для перевірки поведінки системи при отриманні некоректних або неочікуваних даних [7].

Основними сценаріями негативного тестування є:

– відсутність обов’язкових полів;

– некоректний формат JSON;

– неправильний тип даних;

– помилкові параметри авторизації;

– запити до неіснуючих ресурсів.

Очікуваний результат статус-код 400 Bad Request або повідомлення про помилку валідації.

Негативне тестування дозволяє оцінити стійкість API до помилкових дій користувача та забезпечує підвищення надійності вебзастосунку.

1.3.3 Тестування HTTP-статусів

Одним із ключових аспектів API-тестування є перевірка HTTP-статусів. Статус-коди дозволяють визначити результат виконання HTTP-запиту та є важливою частиною REST API [4].

Автоматизовані тести перевіряють:

– відповідність статус-кодів типу запиту;

– правильність обробки помилок;

– стабільність роботи сервера.

Приклади перевірок:

– 200 OK – успішне отримання ресурсу;

– 201 Created – успішне створення;

– 401 Unauthorized – помилка авторизації;

– 404 Not Found – ресурс не знайдено;

– 500 Internal Server Error – помилка сервера.

Приклад перевірки статус-коду у Postman: pm.response.to.have.status(200).

1.3.4 Тестування структури JSON-відповідей

REST API найчастіше використовує формат JSON для передачі даних між сервером і клієнтом. Тому важливим етапом тестування є перевірка структури JSON-відповідей [7].

Автоматизовані перевірки дозволяють:

– контролювати наявність обов’язкових полів;

– перевіряти типи даних;

– аналізувати вкладені об’єкти;

– виявляти порушення структури API.

Такі перевірки є важливими для забезпечення стабільності клієнтських застосунків, які працюють із REST API.

1.3.5 Тестування авторизації та безпеки

Сучасні REST API часто використовують механізми авторизації та аутентифікації:

– JWT;

– OAuth 2.0;

– API Keys;

– Bearer Tokens 7].

Тестування безпеки API включає:

– перевірку доступу без токена;

– перевірку прострочених токенів;

– тестування недостатніх прав доступу;

– перевірку захищених ендпоінтів.

Приклад перевірки авторизації:

– надсилання GET-запиту без токена;

– перевірка отримання статус-коду 401 Unauthorized.

Такі тести дозволяють виявляти потенційні проблеми безпеки вебзастосунків.

1.3.6 Автоматизація тестування у CI/CD

Однією з ключових переваг автоматизованого тестування API є можливість інтеграції тестів у CI/CD-процеси [7].

Автоматизовані API-тести можуть запускатися:

– після кожного коміту;

– перед релізом;

– при оновленні серверної логіки;

– під час регресійного тестування.

Для автоматизації запуску тестів у роботі використовується Newman, який дозволяє виконувати колекції тестів через командний рядок та генерувати HTML-звіти.

Інтеграція тестування у CI/CD дозволяє:

– скоротити час перевірки API;

– швидше виявляти помилки;

– підтримувати стабільність системи;

– автоматизувати контроль якості програмного забезпечення.

Таким чином, автоматизоване тестування API є важливою складовою сучасної розробки вебзастосунків. Використання функціональних, негативних, структурних та безпекових тестів дозволяє забезпечити стабільність REST API, скоротити час тестування та підвищити якість програмного забезпечення.

1.4 Аналіз сучасних інструментів тестування REST API

Для автоматизованого тестування REST API використовуються спеціалізовані програмні засоби, які дозволяють виконувати HTTP-запити, перевіряти відповіді сервера, реалізовувати автоматизовані сценарії тестування та формувати звітність. Вибір інструменту тестування є важливим етапом розробки системи автоматизованого тестування, оскільки від нього залежить швидкість створення тестів, зручність роботи та можливість інтеграції у процеси CI/CD [7].

Серед найбільш поширених інструментів тестування REST API можна виділити:

– Postman;
– SoapUI;
– Insomnia;
– Swagger;
– REST Assured.

Кожен із цих інструментів має власні особливості, переваги та недоліки.

1.4.1 Аналіз можливостей Postman

Одним із найпопулярніших інструментів автоматизованого тестування REST API є Postman. Даний інструмент дозволяє створювати HTTP-запити, формувати колекції тестів, використовувати змінні

середовища, реалізовувати автоматизовані перевірки за допомогою JavaScript, виконувати тестування авторизації, генерувати звіти [7].

Основними перевагами Postman є:

– графічний інтерфейс користувача;
– простота освоєння;
– підтримка REST API;
– можливість автоматизації;
– інтеграція з CI/CD через Newman;
– велика кількість навчальних матеріалів.

Postman підтримує: GET; POST; PUT; PATCH; DELETE-запити; Bearer Token; OAuth 2.0; API Key; JWT авторизацію.

Також Postman дозволяє працювати із JSON-відповідями та створювати складні сценарії тестування.

1.4.2 Аналіз можливостей SoapUI

SoapUI є популярним інструментом для тестування SOAP та REST API. Він підтримує функціональне тестування, навантажувальне тестування, тестування безпеки, автоматизацію сценаріїв [7].

Основними перевагами SoapUI є:

– підтримка SOAP;
– розширені можливості тестування;
– підтримка Groovy-скриптів;
– інтеграція з CI/CD.

Недоліками є:

– складніший інтерфейс;
– більші вимоги до ресурсів;
– складніше налаштування для початківців.

У даній роботі SoapUI не використовується через вищу складність та орієнтацію переважно на enterprise-рішення.

1.4.3Аналіз можливостей Insomnia

Insomnia є сучасним REST API клієнтом із підтримкою HTTP-запитів, GraphQL, змінних середовища, автоматизації [7].

Основними перевагами Insomnia є:

– мінімалістичний інтерфейс;

– простота роботи;

– підтримка плагінів;

– швидкість роботи.

Недоліками є:

– менша популярність;

– обмежені можливості автоматизації;

– менша кількість прикладів та документації у порівнянні з Postman.

1.4.4 Аналіз Swagger та REST Assured

Swagger використовується переважно для документування API, тестування ендпоінтів, генерації OpenAPI-специфікацій [7].

Swagger є важливим інструментом для ознайомлення зі структурою REST API, проте його можливості автоматизації тестування є обмеженими.

REST Assured є Java-бібліотекою для автоматизованого тестування REST API.

REST Assured дозволяє:

– реалізовувати складні тестові сценарії;

– інтегрувати API-тести у Java-проєкти;

– працювати з TestNG та JUnit.

Недоліками REST Assured є:

– необхідність знання Java;

– складніша реалізація;

– відсутність графічного інтерфейсу.

1.4.5 Порівняння інструментів тестування REST API

Порівняльний аналіз сучасних інструментів API-тестування наведено у таблиці 1.3.

Таблиця 1.3 – Порівняння інструментів тестування REST API

Інструмент	GUI	Автоматизація	Мова скриптів	CI/CD	Простота використання
Postman	Так	Так	JavaScript	Так	Висока
SoapUI	Так	Так	Groovy	Так	Середня
Insomnia	Так	Частково	JavaScript	Частково	Висока
Swagger	Так	Обмежено	Немає	Частково	Висока
REST Assured	Ні	Так	Java	Так	Середня

Після аналізу інструментів було визначено, що найбільш доцільним для реалізації програмної системи автоматизованого тестування у межах даної роботи є використання Postman та Newman.

Вибір обумовлений:

– простотою використання;
– підтримкою автоматизації;
– можливістю інтеграції з CI/CD;
– підтримкою JavaScript;
– широким використанням у сучасних QA-командах;
– наявністю інструментів генерації звітності [8].

Таким чином, сучасні інструменти автоматизованого тестування REST API дозволяють реалізовувати комплексні сценарії перевірки вебзастосунків. Використання Postman та Newman забезпечує швидке створення тестів, автоматизацію запуску перевірок та формування звітності про результати тестування.

1.5 Висновки за розділом 1

У першому розділі було проаналізовано предметну область автоматизованого тестування REST API вебзастосунків та досліджено основні технології, що використовуються для взаємодії клієнтської та серверної частин сучасних вебсистем.

Розглянуто особливості HTTP-протоколу та REST API, які є основою більшості сучасних вебзастосунків. Проаналізовано структуру HTTP-запитів і відповідей, основні HTTP-методи та статус-коди, а також принципи побудови REST API.

Досліджено CRUD-операції як базовий механізм роботи з ресурсами у REST API. Розглянуто особливості реалізації операцій Create, Read, Update та Delete, а також їх роль у процесі автоматизованого тестування вебзастосунків.

Проаналізовано основні методи автоматизованого тестування API, зокрема функціональне тестування, негативне тестування, перевірку HTTP-статусів, тестування структури JSON-відповідей та механізмів авторизації. Визначено переваги автоматизованого тестування у порівнянні з ручним тестуванням.

Проведено аналіз сучасних інструментів тестування REST API, серед яких розглянуто Postman, SoapUI, Insomnia, Swagger та REST Assured. За результатами порівняльного аналізу обґрунтовано вибір Postman та Newman як основних засобів реалізації програмної системи автоматизованого тестування у межах даної роботи.

Отримані результати аналізу предметної області та сучасних методів тестування створюють основу для подальшої розробки програмної системи автоматизованого тестування вебзастосунків на основі аналізу HTTP-запитів.

2 МЕТОДИ ТА ЗАСОБИ АНАЛІЗУ HTTP-ЗАПИТІВ

2.1 Структура HTTP-запитів та відповідей

HTTP-протокол є основою взаємодії між клієнтською та серверною частинами вебзастосунків. У процесі роботи REST API клієнт надсилає HTTP-запити, а сервер повертає HTTP-відповіді. Аналіз структури таких повідомлень є важливим етапом автоматизованого тестування вебзастосунків [9].

HTTP-запит складається з стартового рядка, HTTP-заголовків, тіла запиту.

Стартовий рядок містить:

– HTTP-метод;

– URI ресурсу;

– версію HTTP-протоколу.

HTTP-заголовки містять службову інформацію про запит або відповідь. Найпоширеніші заголовки наведено у таблиці 2.1.

Таблиця 2.1 – Основні HTTP-заголовки

Заголовок	Призначення
Content-Type	Формат переданих даних
Authorization	Дані авторизації
Accept	Очікуваний формат відповіді
User-Agent	Інформація про клієнт
Cookie	Дані сесії користувача

Тіло HTTP-запиту використовується переважно у методах POST, PUT та PATCH і містить дані, які передаються серверу. У REST API найчастіше використовується формат JSON [9].

HTTP-відповідь сервера складається з статусного рядка, HTTP-заголовків, тіла відповіді.

Статусний рядок містить:

– версію HTTP;

– статус-код;

– текстове пояснення.

Статус-коди поділяються на такі групи:

– 1xx – інформаційні;

– 2xx – успішні;

– 3xx – перенаправлення;

– 4xx – помилки клієнта;

– 5xx – помилки сервера [9].

Правильний аналіз HTTP-запитів і відповідей дозволяє:

– досліджувати логіку роботи REST API;

– виявляти помилки передачі даних;

– перевіряти авторизацію;

– автоматизувати тестування;

– аналізувати стабільність вебзастосунків.

2.2 Методи аналізу HTTP-запитів у DevTools

Сучасні браузери надають вбудовані інструменти розробника (Developer Tools), які дозволяють аналізувати HTTP-трафік вебзастосунків. Одним із найбільш популярних інструментів є Chrome DevTools [10].

Chrome DevTools дозволяє:

– переглядати HTTP-запити;

– аналізувати заголовки;

– переглядати JSON-відповіді;

– визначати час виконання запитів;

– аналізувати авторизацію;

– досліджувати взаємодію клієнта та сервера.

Для аналізу REST API використовується вкладка Network.

Основними типами мережевих запитів є:

– Fetch/XHR;

– Document;

– CSS;

– JS;

– Media.

Для аналізу REST API найбільш важливими є Fetch/XHR-запити, оскільки саме вони використовуються для передачі HTTP-запитів між клієнтом і сервером [10].

Процес аналізу HTTP-запитів включає відкриття DevTools, перехід до вкладки Network, фільтрацію Fetch/XHR, виконання дій у вебзастосунку та аналіз HTTP-запитів.

Під час аналізу HTTP-запитів досліджуються:

– URL запиту;

– HTTP-метод;

– заголовки;

– параметри;

– тіло запиту;

– JSON-відповідь;

– статус-код;

– час виконання.

На рисунку 2.1 наведено загальну схему аналізу HTTP-запитів у Chrome DevTools.

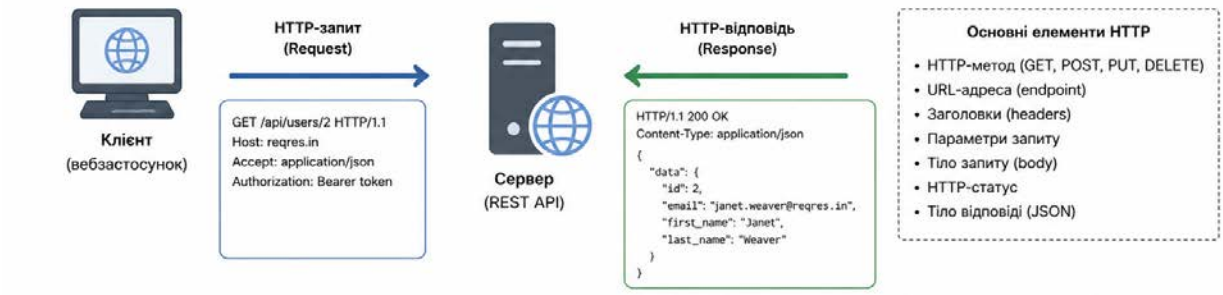


Рисунок 2.1 – Загальна схема аналізу HTTP-запитів

Однією з важливих можливостей Chrome DevTools є функція Copy as cURL, яка дозволяє експортувати HTTP-запит для подальшого використання у системах автоматизованого тестування [11].

Отримані HTTP-запити можуть бути імпортовані до Postman для створення автоматизованих тестових сценаріїв.

Аналіз HTTP-запитів через DevTools дозволяє:

– досліджувати REST API без документації;
– швидко створювати API-тести;
– визначати структуру авторизації;
– аналізувати помилки серверної логіки;
– виконувати тестування вебзастосунків.

Таким чином, Chrome DevTools є важливим інструментом для аналізу HTTP-запитів та побудови систем автоматизованого тестування REST API.

2.3 Архітектура систем автоматизованого тестування

Система автоматизованого тестування REST API є комплексом програмних компонентів, які забезпечують виконання HTTP-запитів, перевірку відповідей сервера та формування результатів тестування [11].

Основною метою такої системи є:

– автоматизація перевірок;
– скорочення часу тестування;
– підвищення стабільності API;
– зменшення кількості ручних операцій.

Архітектура системи автоматизованого тестування складається з таких компонентів:

– модуль аналізу HTTP-запитів;
– модуль створення тестів;
– модуль виконання тестів;
– модуль перевірки результатів;

– модуль формування звітності.

Загальна архітектура системи наведена на рисунку 2.2.

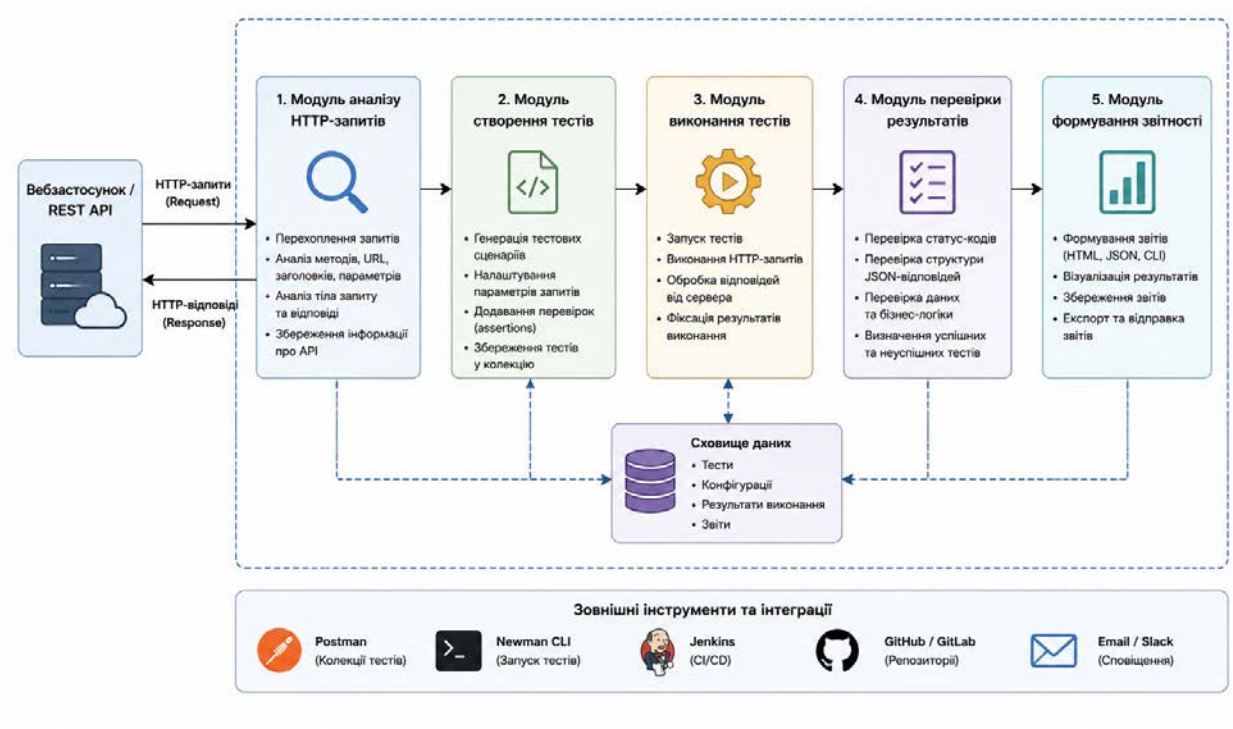


Рисунок 2.2 – Загальна архітектура системи

Модуль аналізу HTTP-запитів відповідає за:

– дослідження REST API;

– визначення URI;

– аналіз HTTP-методів;

– аналіз заголовків та параметрів.

Модуль створення тестів реалізується у середовищі Postman та дозволяє:

– створювати колекції запитів;

– використовувати змінні середовища;

– реалізовувати JavaScript-перевірки.

Модуль виконання тестів реалізується через Newman, який забезпечує:

– автоматичний запуск тестів;

– пакетне виконання сценаріїв;

– інтеграцію з CI/CD;

– генерацію звітності.

Модуль перевірки результатів виконує:

– перевірку HTTP-статусів;

– аналіз JSON-відповідей;

– перевірку часу відповіді;

– аналіз коректності структури даних.

Модуль звітності дозволяє:

– формувати HTML-звіти;

– відображати статистику тестів;

– аналізувати успішність перевірок;

– зберігати результати тестування.

Використання модульної архітектури дозволяє:

– масштабувати систему;

– додавати нові тестові сценарії;

– повторно використовувати колекції тестів;

– інтегрувати систему у CI/CD-процеси .

Таким чином, архітектура систем автоматизованого тестування забезпечує гнучкість, масштабованість та можливість автоматизованої перевірки REST API вебзастосунків.

2.4 UML-моделювання програмної системи

Для опису структури та логіки роботи програмної системи автоматизованого тестування використовується UML (Unified Modeling Language). UML дозволяє візуалізувати компоненти системи, взаємодію між ними та основні сценарії роботи [12].

У межах даної роботи використовуються:

– Use Case Diagram;

– Sequence Diagram;

– Component Diagram;

– Activity Diagram.

2.4.1 Use Case Diagram

Use Case Diagram описує взаємодію користувача із системою автоматизованого тестування.

Основними акторами системи є:

– QA Engineer;

– REST API Server.

Основні варіанти використання:

– аналіз HTTP-запитів;

– створення API-тестів;

– запуск тестування;

– перевірка результатів;

– генерація звітів.

Загальна схема Use Case Diagram наведена на рисунку 2.3.

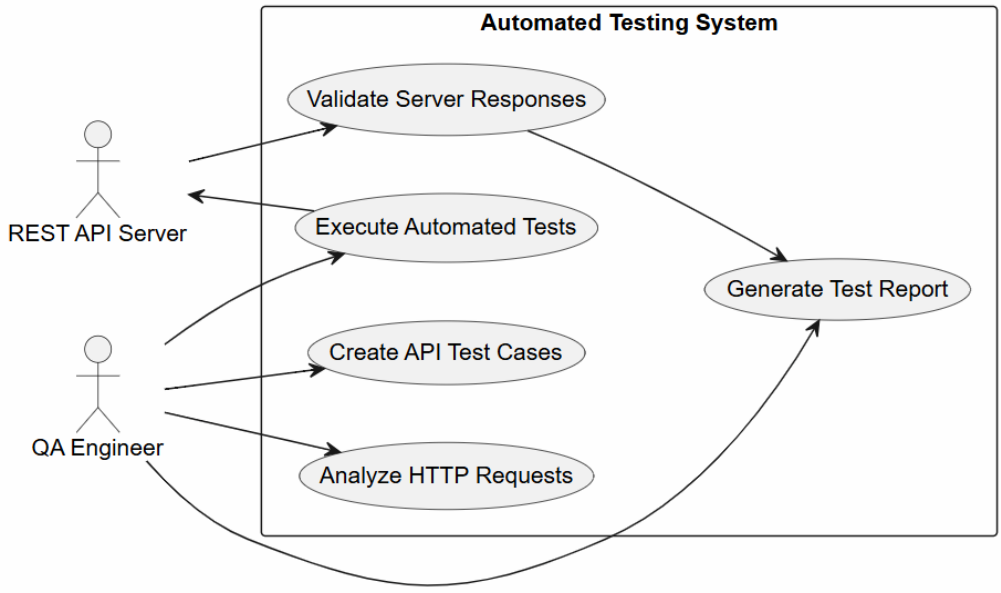


Рисунок 2.3 – Загальна схема Use Case Diagram

2.4.2 Sequence Diagram

Sequence Diagram описує послідовність взаємодії між компонентами системи.

Послідовність роботи системи:

– користувач запускає тест;

– Postman виконує HTTP-запит;

– REST API повертає відповідь;

– система перевіряє результат;

– Newman формує звіт.

Схема Sequence Diagram наведена на рисунку 2.4.

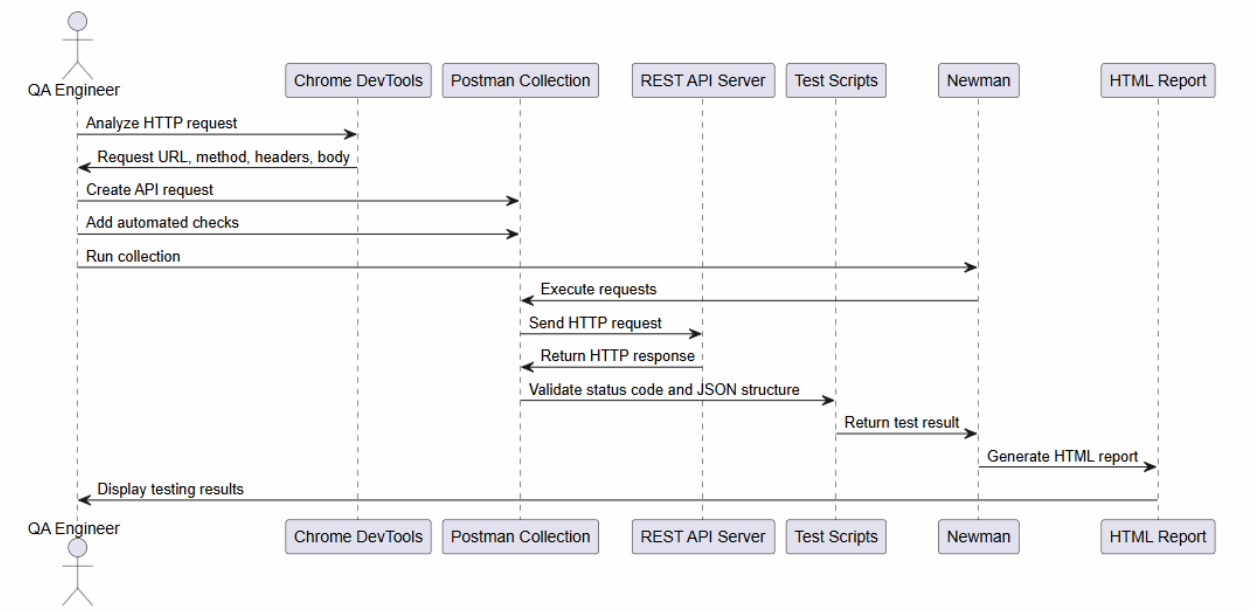


Рисунок 2.4 – Схема Sequence Diagram

2.4.3 Component Diagram

Component Diagram описує структуру програмної системи та її основні компоненти.

Основними компонентами є:

– Chrome DevTools;

– Postman Collection;

– JavaScript Tests;

– Newman Runner;

– HTML Reports.

Схема компонентів наведена на рисунку 2.5.

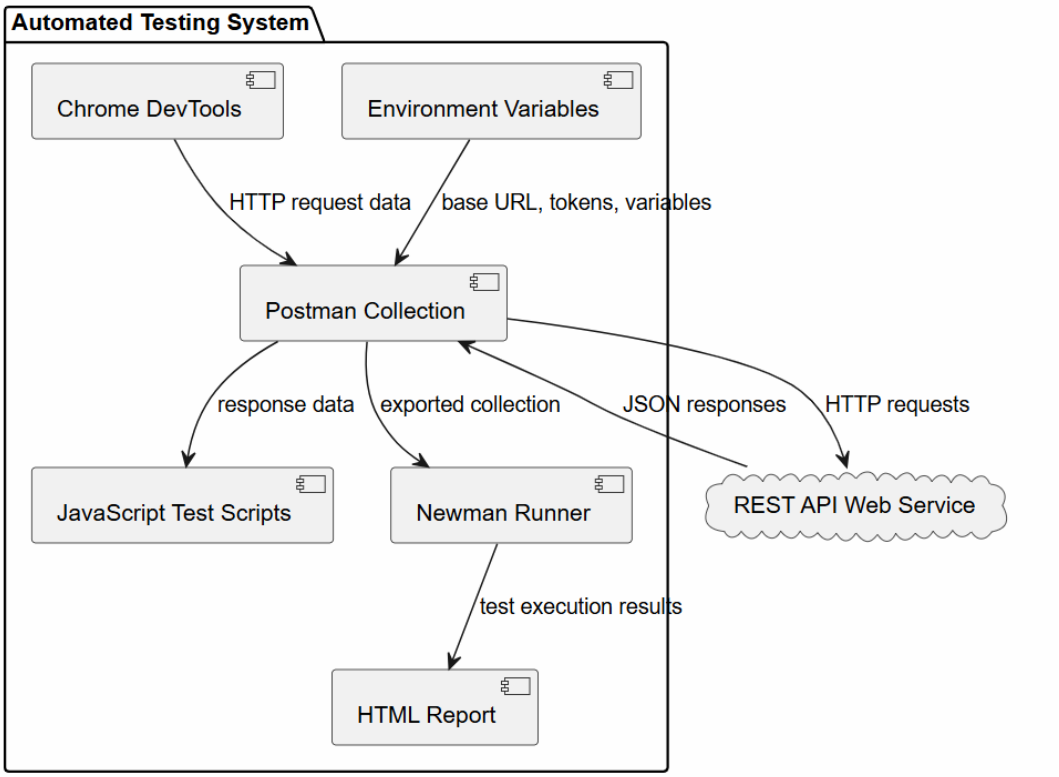


Рисунок 2.5 – Схема компонентів

2.4.4 Activity Diagram

Activity Diagram відображає алгоритм роботи системи автоматизованого тестування.

Основні етапи:

– аналіз HTTP-запиту;

– створення тесту;

– виконання HTTP-запиту;

– перевірка відповіді;

– формування звіту.

Схема Activity Diagram наведена на рисунку 2.6.

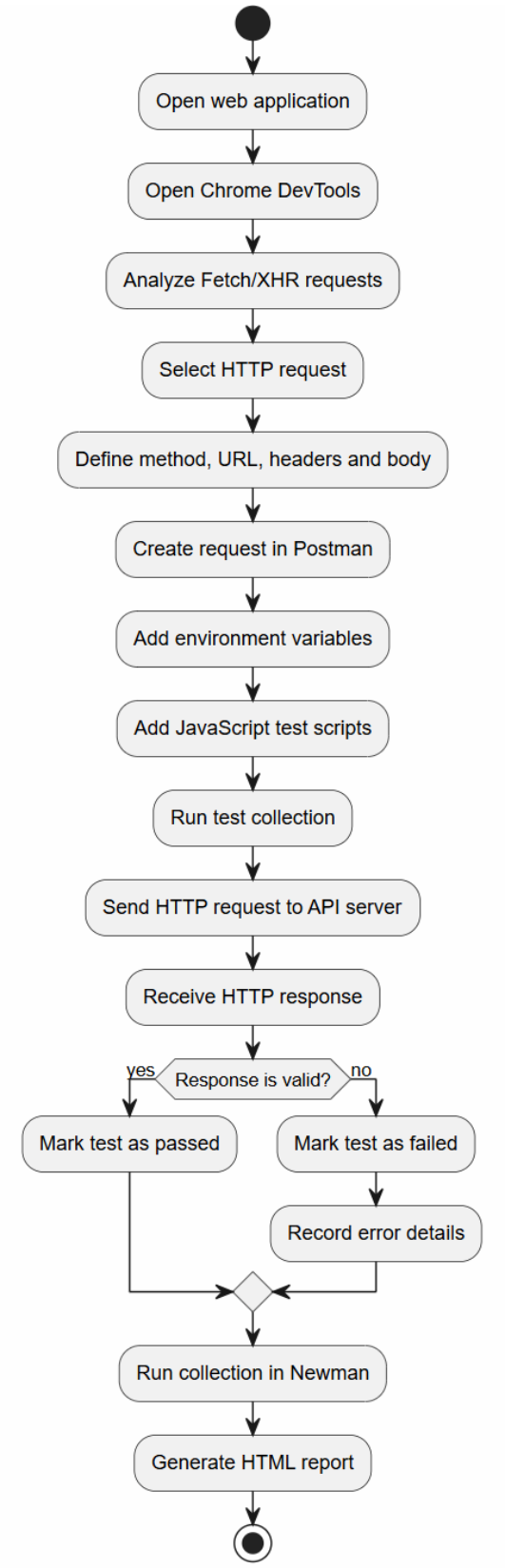


Рисунок 2.6 – Схема Activity Diagram

Таким чином, UML-моделювання дозволяє формалізувати структуру програмної системи автоматизованого тестування та спростити процес її проєктування.

2.5 Висновки за розділом 2

У другому розділі було досліджено методи та засоби аналізу HTTP-запитів у сучасних вебзастосунках.

Проаналізовано структуру HTTP-запитів і відповідей, основні HTTP-заголовки, методи та статус-коди. Визначено особливості передачі даних у REST API із використанням формату JSON.

Досліджено можливості Chrome DevTools для аналізу HTTP-трафіку вебзастосунків. Визначено основні етапи аналізу Fetch/XHR-запитів та способи отримання параметрів REST API для подальшої автоматизації тестування.

Розглянуто архітектуру систем автоматизованого тестування REST API, яка включає модулі аналізу HTTP-запитів, створення тестів, виконання перевірок та формування звітності. Обґрунтовано використання Postman та Newman як основних компонентів програмної системи.

Виконано UML-моделювання програмної системи автоматизованого тестування, що дозволило формалізувати структуру системи та основні сценарії її роботи.

Отримані результати створюють основу для практичної реалізації програмної системи автоматизованого тестування REST API вебзастосунків, яка буде розроблена у наступному розділі.

3 РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ ТЕСТУВАННЯ

3.1 Структура програмної системи

Програмна система автоматизованого тестування вебзастосунків на основі аналізу HTTP-запитів призначена для скорочення часу перевірки REST API, зменшення кількості ручних операцій та автоматичного контролю коректності відповідей сервера.

У межах роботи система складається з таких основних компонентів:

– модуля аналізу HTTP-запитів;
– модуля створення тестових запитів;
– модуля автоматизованих перевірок;
– модуля запуску тестів;
– модуля формування звітності.

Загальна структура програмної системи наведена на рисунку 3.1.

Рисунок 3.1 – Структура програмної системи автоматизованого тестування

На першому етапі виконується аналіз HTTP-запитів вебзастосунку за допомогою Chrome DevTools. Це дозволяє визначити адресу ендпоінта,

HTTP-метод, заголовки, параметри запиту, тіло запиту та структуру відповіді сервера.

На другому етапі отримані HTTP-запити відтворюються у середовищі Postman. Для цього створюється колекція запитів, яка містить набір тестових сценаріїв для перевірки REST API.

На третьому етапі до кожного запиту додаються автоматизовані перевірки, реалізовані мовою JavaScript. Вони дозволяють перевіряти статус-коди, структуру JSON-відповідей, наявність обов’язкових полів, типи даних та час відповіді сервера.

На четвертому етапі колекція тестів запускається за допомогою Newman. Це дозволяє виконувати тестування через командний рядок та формувати звіти про результати перевірок.

Для практичної реалізації системи використано тестові REST API-сервіси ReqRes та Swagger Petstore. Вони дозволяють виконувати типові HTTP-запити, що відповідають CRUD-операціям, і є зручними для демонстрації роботи програмної системи.

3.2 Реалізація API-тестів у Postman

Для реалізації автоматизованих API-тестів було використано середовище Postman. У межах роботи створено колекцію запитів, яка містить основні сценарії перевірки REST API вебзастосунку.

Колекція включає такі групи запитів:

- отримання списку користувачів;
- отримання користувача за ідентифікатором;
- створення нового користувача;
- оновлення даних користувача;
- видалення користувача;
- перевірка обробки помилкових запитів.

Для тестування використовувалися такі ендпоінти:

– GET https://reqres.in/api/users?page=2;

– GET https://reqres.in/api/users/2;

– POST https://reqres.in/api/users;

– PUT https://reqres.in/api/users/2;

– PATCH https://reqres.in/api/users/2;

– DELETE https://reqres.in/api/users/2;

У Postman для кожного запиту було створено окремий тестовий сценарій. Наприклад, для перевірки отримання користувача за ідентифікатором використовується GET-запит GET /api/users/2.

Очікувана відповідь сервера має містити статус-код 200 OK та JSON-об’єкт з інформацією про користувача.

Приклад автоматизованої перевірки статус-коду pm.response.to.have.status(200).

Для перевірки структури JSON-відповіді використовується такий скрипт як зображено на рисунку 3.2

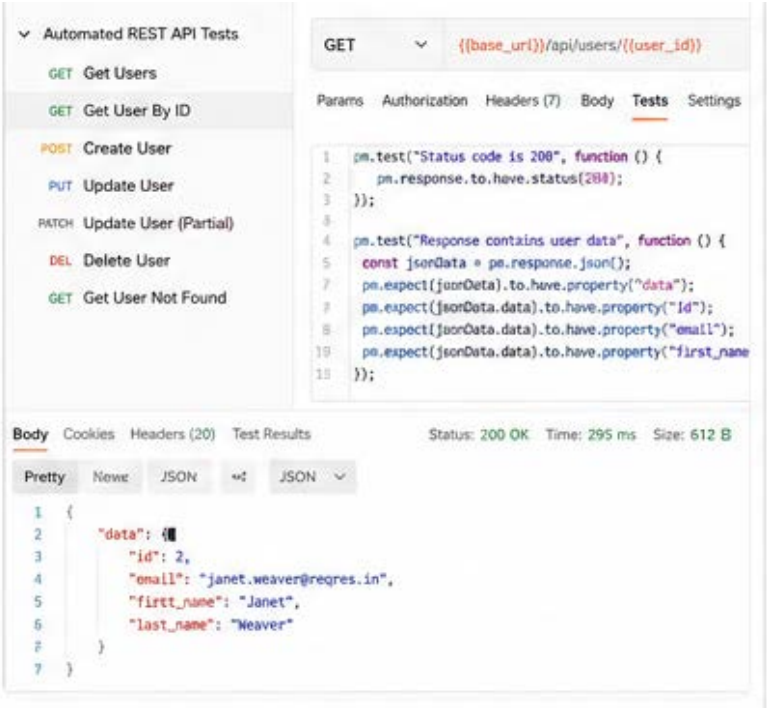


Рисунок 3.2 – Структура перевірки даних користувача у Postman

Для перевірки типів даних використовується фрагмент програми наведений у додатку А.

Під час створення нового користувача використовується POST-запит POST /api/users. Тіло запиту: { "name": "Olena", "job": "QA Engineer"}.

Для цього запиту реалізовано перевірку статус-коду 201 Created: pm.response.to.have.status(201).

Також перевіряється наявність створеного об’єкта у відповіді (додаток А).

Для оновлення користувача застосовано PUT- та PATCH-запити.

Приклад перевірки оновлення даних наведено в додатку А.

Для видалення користувача використано DELETE-запит DELETE /api/users/2.

Очікуваний результат – статус-код 204 No Content pm.response.to.have.status(204).

Окремо реалізовано негативний тест (рис.3.3) для перевірки неіснуючого користувача: GET /api/users/23

Очікуваний результат – статус-код 404 Not Found pm.response.to.have.status(404).

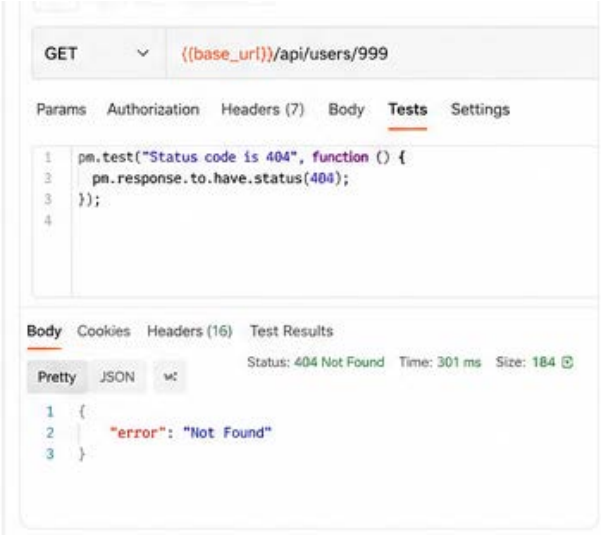


Рисунок 3.3 – Негативний тест у Postman

Таким чином, у Postman було реалізовано набір автоматизованих API-тестів, які перевіряють основні сценарії взаємодії з REST API.

3.3 Автоматизація запуску тестів за допомогою Newman

Для автоматичного запуску тестів використано Newman – інструмент командного рядка, який дозволяє виконувати Postman-колекції без відкриття графічного інтерфейсу Postman.

Перед запуском тестів колекцію Postman було експортовано у формат JSON. Також за потреби експортується файл середовища, у якому зберігаються змінні base_url = https://reqres.in, user_id = 2.

Приклад використання змінної у Postman {{base_url}}/api/users/{{user_id}}.

Такий підхід дозволяє легко змінювати базову адресу API або параметри тестування без редагування кожного запиту окремо.

Для запуску тестів через Newman використовується команда newman run automated_api_tests_collection.json.

Якщо використовується файл середовища, команда має такий вигляд newman run automated_api_tests_collection.json -e environment.json.

Для генерації HTML-звіту використовується команда newman run automated_api_tests_collection.json -e environment.json -r cli,html --reporter-html-export report.html.

Після виконання команди Newman запускає всі запити з колекції, виконує JavaScript-перевірки та формує звіт про результати тестування.

У звіті відображаються:

- кількість виконаних запитів;
- кількість успішних перевірок;
- кількість невдалих перевірок;
- час виконання тестів;
- деталі помилок;

– статистика за кожним запитом.

Автоматизація запуску через Newman дозволяє використовувати створену систему не лише локально, а й у процесах безперервної інтеграції. Наприклад, тести можуть запускатися після кожного оновлення коду або перед релізом нової версії вебзастосунку.

3.4 Генерація звітності та аналіз результатів

Важливим елементом програмної системи автоматизованого тестування є формування звітності. Звіт дозволяє оцінити результати виконання тестів, виявити помилки та проаналізувати стабільність REST API.

У межах роботи звітність формується за допомогою Newman. HTML-звіт містить інформацію про кожен виконаний запит та результат кожної перевірки. Приклад структури звіту зображено на рисунку 3.4.

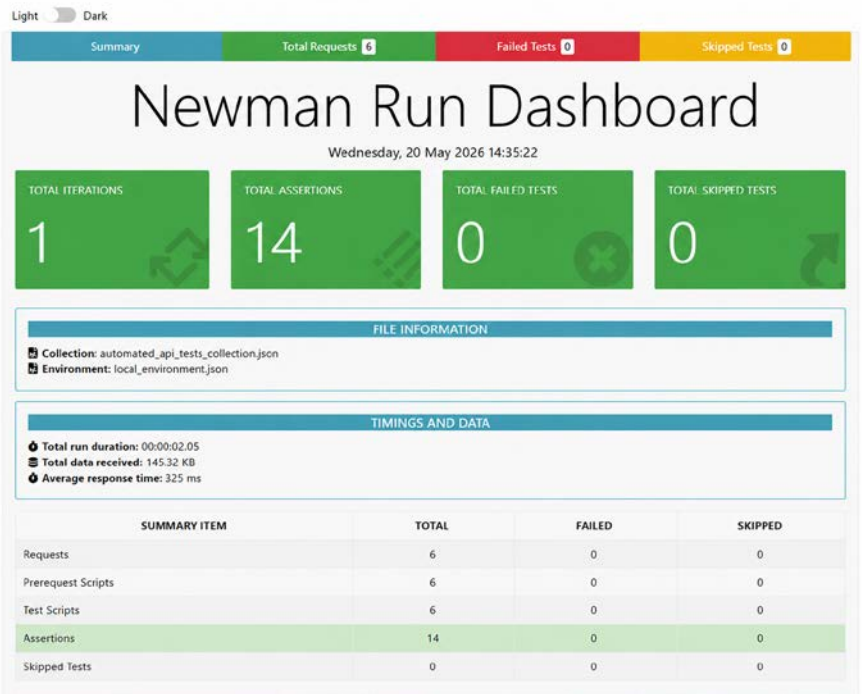


Рисунок 3.4 – HTML-звіт з результатами тесту

Основними показниками аналізу результатів є:

– кількість виконаних тестів;

– кількість успішних перевірок;
– кількість помилок;
– середній час відповіді сервера;
– стабільність повторного запуску тестів.

3.5 Висновки за розділом 3

У третьому розділі було розроблено програмну систему автоматизованого тестування вебзастосунків на основі аналізу HTTP-запитів.

Описано загальну структуру системи, яка включає аналіз HTTP-запитів через Chrome DevTools, створення колекції тестів у Postman, реалізацію JavaScript-перевірок, запуск тестів через Newman та формування HTML-звітів.

Реалізовано набір API-тестів для перевірки основних REST API-операцій: отримання даних, створення ресурсу, оновлення ресурсу, видалення ресурсу та обробки помилкових запитів. Для кожного сценарію створено автоматизовані перевірки статус-кодів, структури JSON-відповідей та наявності обов’язкових полів.

Показано спосіб автоматичного запуску тестів через Newman та формування HTML-звіту. Проведено порівняння ручного та автоматизованого тестування, яке показало скорочення часу виконання перевірок.

Розроблена система може використовуватися для регресійного тестування REST API вебзастосунків та бути інтегрована у CI/CD-процеси для регулярного контролю якості програмного забезпечення.

4 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

4.1 Результати автоматизованого тестування

Для перевірки працездатності розробленої програмної системи автоматизованого тестування було проведено експериментальне дослідження REST API вебзастосунків із використанням сервісів:

– ReqRes;

– Swagger Petstore.

Тестування виконувалося за допомогою:

– Postman;

– Newman;

– Chrome DevTools.

У процесі експериментального дослідження було реалізовано та виконано набір автоматизованих тестових сценаріїв для перевірки:

– CRUD-операцій;

– HTTP-статусів;

– структури JSON-відповідей;

– обробки помилок;

– часу відповіді сервера.

Загальна структура тестових сценаріїв наведена у таблиці 4.1.

Таблиця 4.1 – Тестові сценарії REST API

Тестовий сценарій	HTTP-метод	Очікуваний результат
Отримання списку користувачів	GET	Статус 200
Отримання користувача за ID	GET	JSON-відповідь
Створення користувача	POST	Статус 201
Оновлення користувача	PUT/PATCH	Статус 200
Видалення користувача	DELETE	Статус 204
Отримання неіснуючого ресурсу	GET	Статус 404

Для кожного тестового сценарію виконувалися автоматизовані перевірки статус-коду, структури JSON, наявності обов'язкових полів, часу відповіді, стабільності відповіді сервера.

Приклад результату успішного запуску тестів наведено на рисунку 4.1.

Рисунок 4.1 – Результат запуску тестів у Newman

Було виконано 6 HTTP-запитів до API та 14 перевірок (assertions) всередині тестів. Під час тестування було встановлено, що всі реалізовані тестові сценарії успішно виконуються та дозволяють автоматично перевіряти основні REST API-операції.

У результаті проведеного тестування:

– кількість успішних перевірок становила 100%;

– помилок виконання тестів не виявлено;

– середній час відповіді API не перевищував 350 мс.

Отримані результати підтверджують коректність роботи розробленої системи автоматизованого тестування.

4.2 Порівняння ручного та автоматизованого тестування

Для оцінювання доцільності використання автоматизованого тестування було проведено порівняння ручного та автоматизованого підходів до перевірки REST API.

Під час ручного тестування користувач повинен:

– виконати HTTP-запит;

– перевірити статус-код;

– проаналізувати JSON-відповідь;

– перевірити структуру даних;

– зафіксувати результат тестування.

При автоматизованому тестуванні ці дії виконуються автоматично за допомогою Postman та Newman.

Результати порівняння наведено у таблиці 4.2.

Таблиця 4.2 – Порівняння ручного та автоматизованого тестування

Показник	Ручне тестування	Автоматизоване тестування
Кількість тестів	6	6
Час виконання одного тесту	2 хв	10 с
Загальний час тестування	12 хв	1 хв
Імовірність людської помилки	Висока	Низька
Повторне виконання	Обмежене	Автоматичне
Масштабованість	Низька	Висока

Порівняння показало, що автоматизоване тестування значно скорочує час виконання перевірок та мінімізує вплив людського фактора.

Крім того, автоматизація дозволяє:

– повторно використовувати тестові сценарії;

– виконувати регресійне тестування;

– інтегрувати перевірки у CI/CD;

– забезпечувати стабільність API після оновлень.

Таким чином, автоматизоване тестування REST API є більш ефективним для сучасних вебзастосунків у порівнянні з ручним тестуванням.

4.3 Аналіз скорочення часу тестування

Однією з основних цілей розробленої системи є скорочення часу перевірки REST API вебзастосунків.

Для оцінювання скорочення часу тестування було порівняно ручний та автоматизований підходи. При ручному тестуванні кожен запит необхідно виконувати окремо, перевіряти статус-код, аналізувати JSON-відповідь та фіксувати результат. При автоматизованому тестуванні ці дії виконуються системою. Приклад порівняння наведено у таблиці 4.2.

На основі порівняння можна зробити висновок, що автоматизоване тестування дозволяє значно скоротити час перевірки REST API. У наведеному прикладі час тестування зменшився з 12 хвилин до 1 хвилини.

Для наведеного прикладу скорочення часу дорівнює ((12 - 1) / 12) * 100% = 91,7%.

Отже, використання автоматизованої системи тестування дозволило скоротити час перевірки REST API приблизно на 91,7%. Таким чином, експериментальне дослідження підтвердило доцільність використання автоматизованого тестування REST API.

4.4 Перспективи масштабування системи

Розроблена програмна система автоматизованого тестування має модульну структуру, що дозволяє масштабувати її та розширювати функціональні можливості.

Подальший розвиток системи може включати:

– інтеграцію з CI/CD;
– підтримку GraphQL API;
– автоматичну генерацію тестів;
– тестування продуктивності;
– інтеграцію з системами баг-трекінгу;
– автоматичне тестування безпеки REST API.

Одним із перспективних напрямів є інтеграція системи з платформами:

– Jenkins;
– GitHub Actions;
– GitLab.

Це дозволить автоматично запускати API-тести після оновлення коду, контролювати стабільність REST API, забезпечити безперервний контроль якості.

Також перспективним напрямом є використання штучного інтелекту для автоматичної генерації тестових сценаріїв, аналізу логів, прогнозування помилок API.

Модульна архітектура системи дозволяє легко додавати нові типи тестів та масштабувати систему для роботи з великими REST API вебзастосунками.

4.5 Висновки за розділом 4

У четвертому розділі було проведено експериментальне дослідження розробленої програмної системи автоматизованого тестування REST API вебзастосунків.

У процесі дослідження виконано тестування CRUD-операцій, перевірку HTTP-статусів, структури JSON-відповідей та негативних

сценаріїв. Отримані результати підтвердили коректність роботи автоматизованих тестів та стабільність REST API.

Проведено порівняння ручного та автоматизованого тестування. Встановлено, що автоматизація дозволяє значно скоротити час перевірки REST API та мінімізувати вплив людського фактора.

Результати експериментального дослідження показали, що використання розробленої системи дозволило скоротити час тестування приблизно на 91,7%.

Також визначено перспективи масштабування системи, серед яких інтеграція з CI/CD, підтримка додаткових типів API та автоматизація аналізу результатів тестування.

Отримані результати підтверджують доцільність використання програмної системи автоматизованого тестування вебзастосунків на основі аналізу HTTP-запитів для забезпечення якості сучасного програмного забезпечення.

ВИСНОВКИ

У дипломній роботі вирішено актуальне завдання автоматизації тестування REST API вебзастосунків шляхом створення програмної системи автоматизованого тестування на основі аналізу HTTP-запитів.

У процесі виконання роботи було проаналізовано особливості HTTP-протоколу та REST API, які є основою взаємодії клієнтської та серверної частин сучасних вебзастосунків. Досліджено структуру HTTP-запитів і відповідей, CRUD-операції, HTTP-статуси та формат JSON, що використовуються для передачі даних між компонентами системи.

Проведено аналіз сучасних методів автоматизованого тестування REST API та інструментів API-тестування. За результатами порівняльного аналізу обґрунтовано використання Postman та Newman як основних засобів реалізації програмної системи автоматизованого тестування.

У роботі досліджено можливості аналізу HTTP-запитів за допомогою Chrome DevTools. Визначено основні методи отримання інформації про REST API, включаючи аналіз HTTP-методів, заголовків, параметрів запитів та JSON-відповідей.

Було розроблено структуру програмної системи автоматизованого тестування REST API вебзастосунків, яка включає:

– модуль аналізу HTTP-запитів;
– модуль створення API-тестів;
– модуль автоматизованих перевірок;
– модуль запуску тестів;
– модуль формування звітності.

У межах практичної частини реалізовано набір автоматизованих тестів для перевірки CRUD-операцій REST API. Створено тестові сценарії для:

– отримання ресурсів;
– створення нових записів;

– оновлення даних;

– видалення ресурсів;

– перевірки негативних сценаріїв.

Реалізовано автоматизовані перевірки:

– HTTP-статусів;

– структури JSON-відповідей;

– типів даних;

– наявності обов’язкових полів;

– часу відповіді сервера.

Для автоматизації запуску тестів використано Newman, що дозволило виконувати пакетний запуск тестових сценаріїв та формувати HTML-звіти про результати тестування.

У результаті експериментального дослідження проведено порівняння ручного та автоматизованого тестування REST API. Встановлено, що використання розробленої програмної системи дозволяє суттєво скоротити час перевірки REST API вебзастосунків.

У ході дослідження визначено, що:

– час виконання тестування скоротився приблизно на 91,7%;

– зменшився вплив людського фактора;

– підвищилася стабільність повторного запуску тестів;

– спростився процес регресійного тестування REST API.

Таким чином, поставлену мету роботи — зменшення часу виявлення помилок у REST API вебзастосунків шляхом створення програмної системи автоматизованого тестування на основі аналізу HTTP-запитів — досягнуто.

Розроблена система може використовуватися для автоматизованого тестування REST API вебзастосунків у процесах забезпечення якості програмного забезпечення та бути інтегрована у сучасні CI/CD-процеси.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. What is a REST API? Benefits, Uses, Examples. URL: https://www.techtarget.com/searchapparchitecture/definition/RESTful-API (date of access: 17.05.2026).

2. What is API testing?. URL: https://www.postman.com/api-platform/api-testing/ (date of access: 17.05.2026).

3. chrome.devtools.network. URL: https://developer.chrome.com/docs/extensions/reference/api/devtools/network (date of access: 17.05.2026).

4. Overview of HTTP. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Overview (date of access: 17.05.2026).

5. HTTP request methods. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Methods (date of access: 17.05.2026).

6. REST API Tutorial. URL: https://restfulapi.net/ (date of access: 17.05.2026).

7. Roy Fielding – Architectural Styles and the Design of Network-based Software Architectures. URL: https://ics.uci.edu/~fielding/pubs/dissertation/top.htm (date of access: 17.05.2026).

8. JSON Official Documentation. URL: https://www.json.org/json-en.html (date of access: 17.05.2026).

9. Postman API Testing Documentation. URL: https://learning.postman.com/docs/tests-and-scripts/write-scripts/test-scripts/ (date of access: 17.05.2026).

10. Chrome DevTools Network Documentation. URL: https://developer.chrome.com/docs/devtools/network/ (date of access: 17.05.2026).

11. Newman Command Line Collection Runner. URL: https://www.npmjs.com/package/newman (date of access: 17.05.2026).

12. A real backend - for testing and building. UML: https://reqres.in/ (date of access: 17.05.2026).

ДОДАТОК А

Текст програми

А.1 Запит GET Users

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});

pm.test("Response time is less than 1000 ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(1000);
});

pm.test("Response contains data array", function () {
    const jsonData = pm.response.json();
    pm.expect(jsonData).to.have.property("data");
    pm.expect(jsonData.data).to.be.an("array");
});

pm.test("Each user has required fields", function () {
    const jsonData = pm.response.json();

    jsonData.data.forEach(function (user) {
        pm.expect(user).to.have.property("id");
        pm.expect(user).to.have.property("email");
        pm.expect(user).to.have.property("first_name");
        pm.expect(user).to.have.property("last_name");
        pm.expect(user).to.have.property("avatar");
    });
});
```

А.2 Запит GET User By ID

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});

pm.test("Response contains user data", function () {
    const jsonData = pm.response.json();

    pm.expect(jsonData).to.have.property("data");
    pm.expect(jsonData.data).to.have.property("id");
    pm.expect(jsonData.data).to.have.property("email");
    pm.expect(jsonData.data).to.have.property("first_name");
    pm.expect(jsonData.data).to.have.property("last_name");
    pm.expect(jsonData.data).to.have.property("avatar");
});

pm.test("User fields have correct data types", function () {
    const jsonData = pm.response.json();

    pm.expect(jsonData.data.id).to.be.a("number");
    pm.expect(jsonData.data.email).to.be.a("string");
    pm.expect(jsonData.data.first_name).to.be.a("string");
    pm.expect(jsonData.data.last_name).to.be.a("string");
    pm.expect(jsonData.data.avatar).to.be.a("string");
});
```

А.3 Запит CREATE User

```
pm.test("Status code is 201", function () {
    pm.response.to.have.status(201);
});

pm.test("User is created successfully", function () {
    const jsonData = pm.response.json();

    pm.expect(jsonData).to.have.property("name");
    pm.expect(jsonData).to.have.property("job");
    pm.expect(jsonData).to.have.property("id");
    pm.expect(jsonData).to.have.property("createdAt");
});

pm.test("Created user contains correct data", function () {
    const jsonData = pm.response.json();

    pm.expect(jsonData.name).to.eql("Olena");
    pm.expect(jsonData.job).to.eql("QA Engineer");
});
```

Тіло запиту:

```
{
  "name": "Olena",
  "job": "QA Engineer"
}
```

А.4 Запит UPDATE User

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});

pm.test("User is updated successfully", function () {
    const jsonData = pm.response.json();

    pm.expect(jsonData).to.have.property("name");
    pm.expect(jsonData).to.have.property("job");
    pm.expect(jsonData).to.have.property("updatedAt");
});

pm.test("Updated user contains correct data", function () {
    const jsonData = pm.response.json();

    pm.expect(jsonData.name).to.eql("Olena");
    pm.expect(jsonData.job).to.eql("Senior QA Engineer");
});
```

Тіло запиту:

```
{
  "name": "Olena",
  "job": "Senior QA Engineer"
}
```

А.5 Запит PATCH User

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});

pm.test("User is partially updated", function () {
    const jsonData = pm.response.json();

    pm.expect(jsonData).to.have.property("job");
    pm.expect(jsonData).to.have.property("updatedAt");
});

pm.test("Updated job is correct", function () {
    const jsonData = pm.response.json();

    pm.expect(jsonData.job).to.eql("Automation QA Engineer");
});
```

Тіло запиту:

```
{
  "job": "Automation QA Engineer"
}
```

А.6 Запит DELETE User

```
pm.test("Status code is 204", function () {
    pm.response.to.have.status(204);
});

pm.test("Response body is empty", function () {
    pm.expect(pm.response.text()).to.eql("");
});
```

А.7 Негативний тест GET User Not Found

```
pm.test("Status code is 404", function () {
    pm.response.to.have.status(404);
});

pm.test("Response body is empty object", function () {
    const jsonData = pm.response.json();

    pm.expect(jsonData).to.be.an("object");
    pm.expect(Object.keys(jsonData).length).to.eql(0);
});
```

А.8 Pre-request Script для генерації тестових даних

```
const randomNumber = Math.floor(Math.random() * 10000);
```

```
pm.environment.set("random_name", "User_" + randomNumber);
pm.environment.set("random_job", "QA_" + randomNumber);
```

A.9 Запит CREATE User з динамічними даними

```
pm.test("Status code is 201", function () {
    pm.response.to.have.status(201);
});

pm.test("Dynamic user is created", function () {
    const jsonData = pm.response.json();

    pm.expect(jsonData.name).to.eql(pm.environment.get("random_name"));
    pm.expect(jsonData.job).to.eql(pm.environment.get("random_job"));
    pm.expect(jsonData).to.have.property("id");
    pm.expect(jsonData).to.have.property("createdAt");
});
```

Тіло запиту:

```
{
  "name": "{{random_name}}",
  "job": "{{random_job}}"
}
```

A.10 Команди запуску колекції через Newman

```
newman run automated_api_tests_collection.json
newman run automated_api_tests_collection.json -e environment.json
newman run automated_api_tests_collection.json -e environment.json -r
cli,html --reporter-html-export report.html
```

ДОДАТОК Б

Слайди презентації

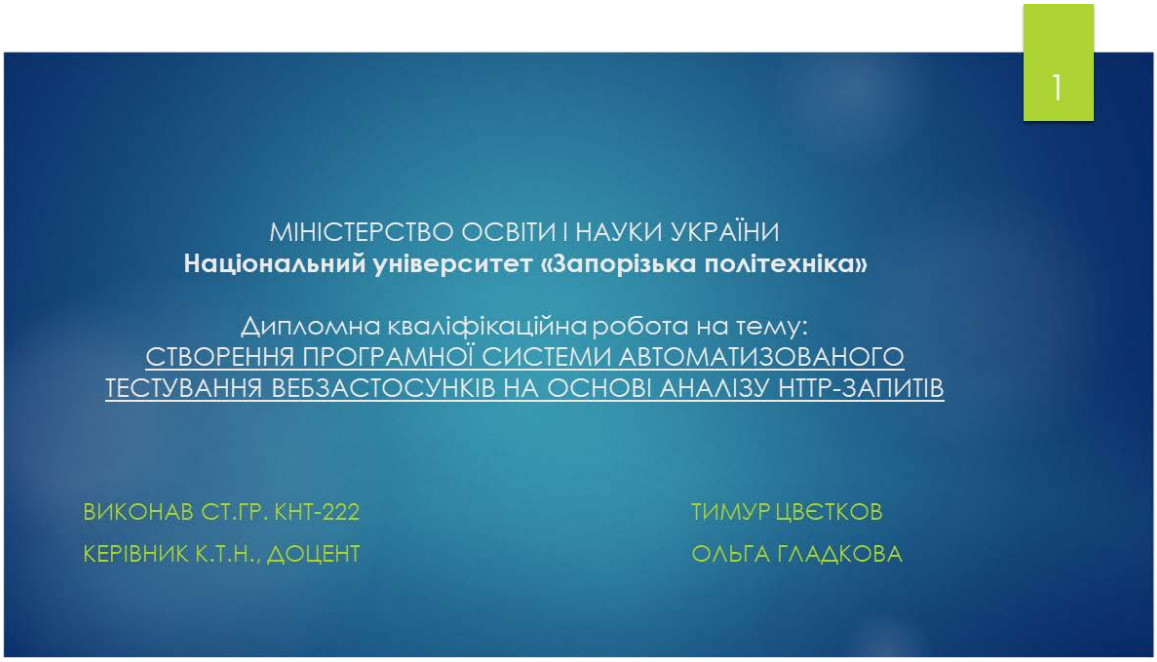


Рисунок Б.1 – Слайд 1

Рисунок Б.2 – Слайд 2

3

Порівняння інструментів тестування REST API

Інструмент	GUI	Автоматизація	Мова скриптів	CI/CD	Простота використання
Postman	Так	Так	JavaScript	Так	Висока
SoapUI	Так	Так	Groovy	Так	Середня
Insomnia	Так	Частково	JavaScript	Частково	Висока
Swagger	Так	Обмежено	Немає	Частково	Висока
REST Assured	Ні	Так	Java	Так	Середня

Рисунок Б.3 – Слайд 3

4

Загальна схема аналізу HTTP-запитів

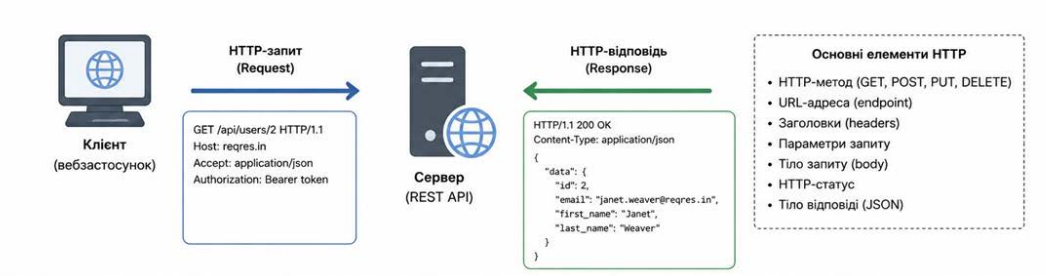


Рисунок Б.4 – Слайд 4

5

Розроблена схема Use Case Diagram

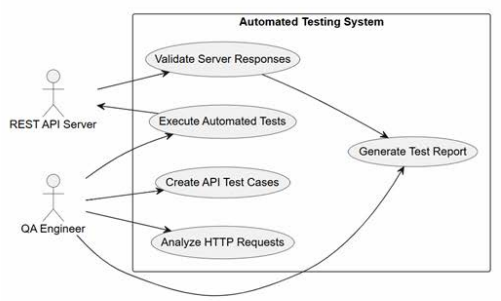
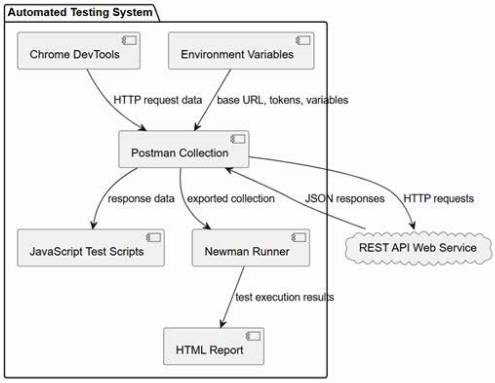


Рисунок Б.5 – Слайд 5

6

Розроблена UML структурна діаграма компонентів системи та Activity Diagram



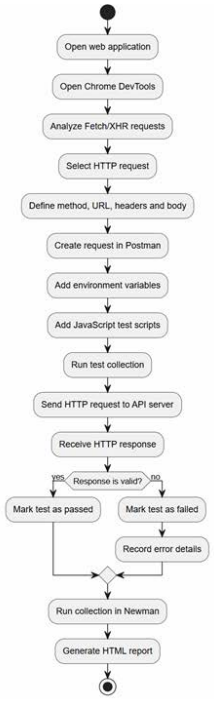


Рисунок Б.6 – Слайд 6

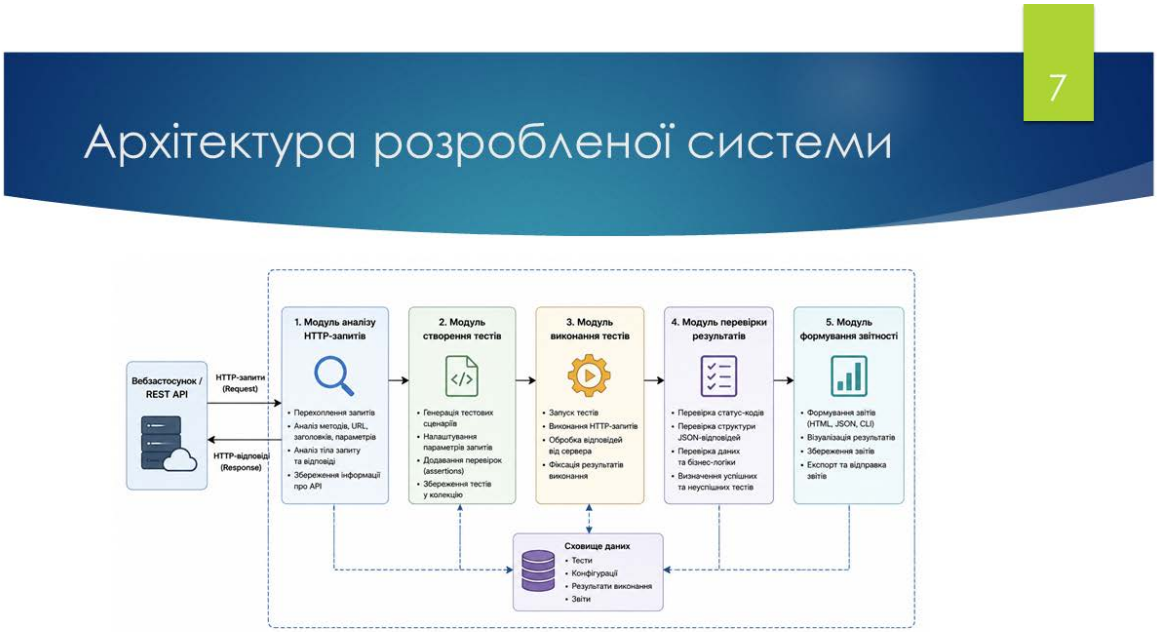


Рисунок Б.7 – Слайд 7

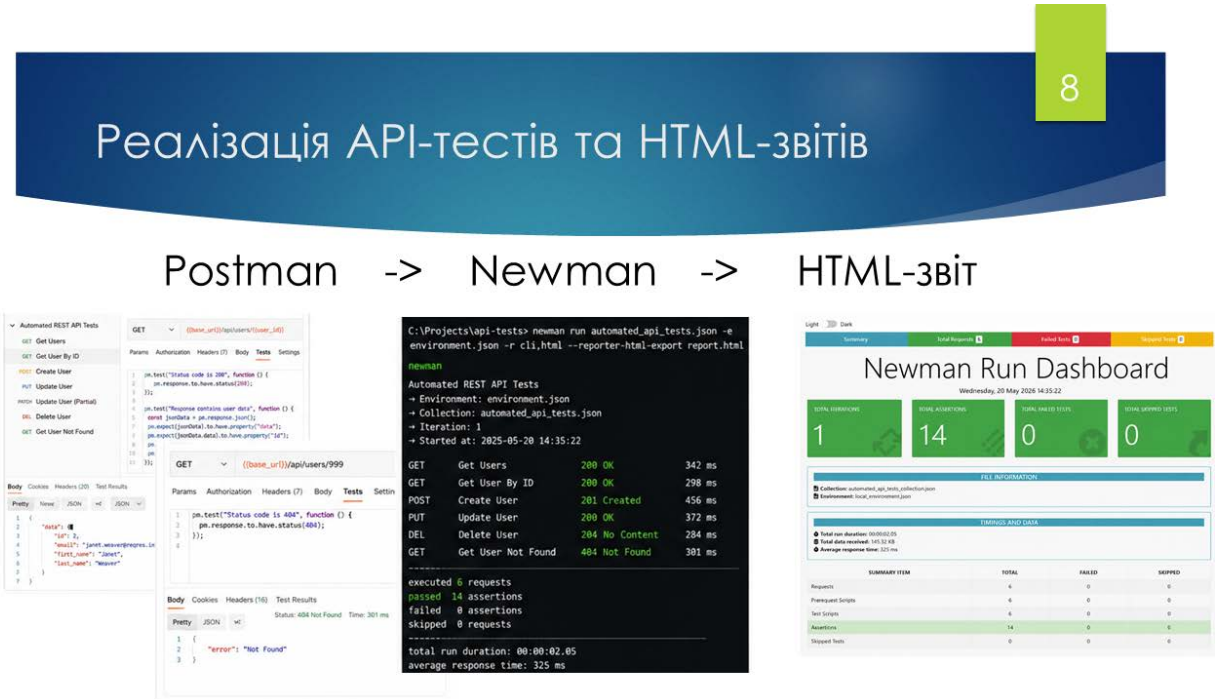


Рисунок Б.8 – Слайд 8

9

Порівняння ручного та автоматизованого тестування

Показник	Ручне тестування	Автоматизоване тестування
Кількість тестів	6	6
Час виконання одного тесту	2 хв	10 с
Загальний час тестування	12 хв	1 хв
Імовірність людської помилки	Висока	Низька
Повторне виконання	Обмежене	Автоматичне
Масштабованість	Низька	Висока

Рисунок Б.9 – Слайд 9

10

Висновки

- Розроблено програмну систему тестування REST API вебзастосунків
- Проаналізовано HTTP-протокол, REST API та CRUD-операції
- Реалізовано API-тести у Postman. Створено автоматизовані перевірки HTTP-статусів та JSON-відповідей
- Реалізовано запуск тестів через Newman.
- Проведено експериментальне дослідження системи, та встановлено скорочення часу тестування приблизно на 91,7%. Підтверджено доцільність використання автоматизованого тестування REST API.

Рисунок Б.10 – Слайд 10