МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Економічний факультет
Кафедра інформаційних технологій та аналітики даних

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавра

на тему: «**Автоматизація тестування додатку адміністрування розкладу занять.»**

Виконав: студент 4 курсу, групи КН-41
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми
«Комп'ютерні науки»
Данило Анатолійович Качмарський

Керівник: *викладач • Навчально-науковий інститут інформаційних технологій та бізнесу*
Сергій Васильович Ляховчук

Рецензент: *кандидат технічних наук, доцент,*
доцент кафедри прикладної математики
Донецького національного університету імені Василя Стуса
Загоруйко Любов Василівна

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ
Завідувач кафедри інформаційних технологій та аналітики даних
_____ (проф., д.е.н. Кривицька О.Р.)
Протокол № ____ від « ___ » _____ 2026 р.

Острог, 2026

АНОТАЦІЯ

кваліфікаційної роботи

на здобуття освітнього ступеня бакалавра

Тема: Тестування (API) серверної частини, веб сторінки для додатку управління та перегляду розкладу та університетської інформації.

Автор: Качмарський Данило Анатолійович

Науковий керівник: викладач Навчально-наукового інституту інформаційних технологій та бізнесу Ляховчук Сергій Васильович.

Захищена «……»…… 20__ року.

Пояснювальна записка до кваліфікаційної роботи: 106 с., 19 рис., 17 табл., 4 додаток, 45 джерел.

Ключові слова: тестування програмного забезпечення, API, FastAPI, Vue.js, Playwright, pytest, автоматизація тестування, доступність, WCAG, CI/CD.

Короткий зміст праці:

Кваліфікаційна робота присвячена проєктуванню та практичній реалізації комплексної системи автоматизованого тестування для багатокомпонентного веб-додатку управління розкладом та університетською інформацією. Об'єктом дослідження є інформаційна система, що включає серверну частину на базі FastAPI з REST API, клієнтський веб-інтерфейс на Vue.js та інтеграційний модуль Telegram-бота.

У роботі розроблено багаторівневу архітектуру тестування за принципом «піраміди тестування», що охоплює модульне (Unit), інтеграційне (Integration), наскрізне (End-to-End), навантажувальне (Performance) та безпекове (Security) тестування. Для автоматизації обрано сучасний інструментарій: pytest для серверної частини, Playwright для веб-інтерфейсу, k6 для оцінки продуктивності, OWASP ZAP для виявлення вразливостей, axe DevTools та NVDA для перевірки доступності за стандартом WCAG 2.1 AA.

Окрему увагу приділено ручному тестуванню доступності веб-інтерфейсу для усіх ключових сторінок системи із застосуванням екранного читача NVDA,

аналізом контрастності кольорів та клавіатурної навігації. За результатами виявлено та задокументовано низку порушень доступності, сформовано рекомендації щодо їх усунення.

Розроблену тестову інфраструктуру інтегровано з GitHub Actions та Docker для безперервної інтеграції. Загальна успішність автоматизованих тестів становить 100%, покриття коду бізнес логіки — понад 75%, наявні файли(які не мають бізнес логіки, що не тестуються). Результати роботи можуть бути використані для забезпечення якості подібних освітніх інформаційних систем.

_____

(підпис автора)

ABSTRACT

of the undergraduate dissertation

for the award of a Bachelor's degree

Topic: Testing (API) of the server-side and web pages for an application to manage and view timetables and university information.

Author: Danylo Anatoliyovych Kachmarskyi

Supervisor: Serhii Vasylovych Lyakhovchuk, Lecturer at the Educational and Research Institute of Information Technologies and Business.

Defended on '……'…… 20__.

Explanatory note to the qualification work: 106 pp., 19 figs., 17 tables, 4 appendix, 45 references.

Keywords: software testing, API, FastAPI, Vue.js, Playwright, pytest, test automation, accessibility, WCAG, CI/CD.

Summary of the thesis:

This thesis is devoted to the design and practical implementation of a comprehensive automated testing system for a multi-component web application for managing timetables and university information. The subject of the study is an information system comprising a server-side component based on FastAPI with a REST API, a client-side web interface built with Vue.js, and a Telegram bot integration module.

The thesis develops a multi-level testing architecture based on the 'testing pyramid' principle, covering unit, integration, end-to-end, performance and security testing. Modern tools were selected for automation: pytest for the server-side, Playwright for the web interface, k6 for performance evaluation, OWASP ZAP for vulnerability detection, and axe DevTools and NVDA for accessibility testing against the WCAG 2.1 AA standard.

Particular attention was paid to manual accessibility testing of the web interface for all key pages of the system, using the NVDA screen reader, analysing colour contrast and keyboard navigation. As a result, a number of accessibility issues were identified and documented, and recommendations were formulated for their resolution.

The developed test infrastructure has been integrated with GitHub Actions and Docker for continuous integration. The overall success rate of automated tests is 100%, with code coverage exceeding 75%. The results of this work can be used to ensure the quality of similar educational information systems.

_____

(author’s signature)

ЗМІСТ

ВСТУП

Динамічний розвиток сучасних вебтехнологій та дедалі вищі вимоги до відмовостійкості програмних продуктів зумовлюють необхідність застосування комплексного, багаторівневого підходу до забезпечення якості. Дана кваліфікаційна робота присвячена проєктуванню, детальній розробці та практичному впровадженню цілісної екосистеми автоматизованого тестування для інтегрованого університетського додатка. Об'єкт дослідження являє собою складну багатокомпонентну систему, що включає високопродуктивну серверну частину на базі фреймворку **FastAPI**, динамічний клієнтський веб-інтерфейс, побудований на **Vue.js**, та інтерактивний модуль взаємодії через **Telegram Bot API**. Актуальність роботи підкріплена потребою в синхронізації процесів тестування між різнорідними платформами для створення єдиного надійного інформаційного середовища управління навчальним процесом

Мета роботи: концептуальне обґрунтування та технічна реалізація універсальної методології багаторівневого тестування (Unit, Integration, E2E, Stress Tests(k6)), яка забезпечує безперервний контроль якості програмного коду на всіх етапах його життєвого циклу. Основним завданням є створення автоматизованого інструментарію для виявлення дефектів, мінімізації технічних ризиків при масштабуванні системи, а також оптимізація взаємодії між API, вебсторінками та месенджер-ботом для гарантування цілісності та безпеки університетських даних у межах єдиної програмної архітектури. Ручне тестування вебсторінки для виявлення порушень доступності та конфліктів. Ведення документації та подача в звіті.

Задачі дослідження:

Завдання дослідження включають комплексний аналіз предметної області, проектування інформаційної системи та вибір інструментарію, методів реалізації та тестування програмного продукту.

1. Провести аналіз предметної області шляхом дослідження сучасних підходів до тестування багатокомпонентних веб-додатків, аналізу архітектури системи управління розкладом та університетською інформацією, вивчення вимог до надійності, безпеки та доступності інформаційних систем освітньої сфери, а також огляду існуючих методик тестування API, веб-інтерфейсів та систем месенджер-інтеграції.
2. Виконати проектування інформаційної системи, що передбачає розробку архітектури комплексної системи тестування для багатошарового веб-додатку, проектування структури тестів різних рівнів, включаючи unit, integration, E2E, performance та security тести, визначення стратегії тестування для кожного компонента системи, а саме серверної частини,

веб-інтерфейсу та Telegram бота, а також формування методики оцінки якості та доступності користувацьких інтерфейсів.

3. Здійснити вибір інструментарію, методів реалізації та тестування програмного продукту, який включає обґрунтування вибору інструментів автоматизованого тестування, таких як pytest та Playwright, вибір методів тестування безпеки з використанням OWASP ZAP та Burp Suite, визначення інструментів для перевірки доступності та валідації веб-інтерфейсів, включаючи axe DevTools та Lighthouse, а також адаптацію методологій тестування під специфіку освітніх інформаційних систем.

Об'єктом дослідження є багатокомпонентна інформаційна система управління розкладом та університетською інформацією, що включає серверну частину на базі FastAPI з REST API, веб-інтерфейс на базі Vue.js та систему Telegram бота для альтернативного доступу до функціональності.

Предметом дослідження є інструментальні засоби та методики тестування програмного забезпечення, які включають методи автоматизованого тестування різних рівнів, такі як unit, integration та E2E тести, технології тестування продуктивності та навантаження, методики перевірки безпеки програмного забезпечення, інструменти оцінки доступності та валідації веб-інтерфейсів, а також стратегії тестування інтеграційних компонентів, зокрема Telegram ботів.

РОЗДІЛ І. ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1. Опис предметного середовища(функціональної моделі, процесу діяльності)

Предметним середовищем даної роботи є сфера тестування програмного забезпечення, зокрема тестування багатокомпонентних веб-додатків, що включають серверну частину, веб-інтерфейси та інтеграційні компоненти. Тестування програмного забезпечення є невід'ємною частиною процесу розробки програмних продуктів, спрямованою на виявлення дефектів, перевірку відповідності вимогам та забезпечення якості кінцевого продукту. У сучасних умовах швидкого розвитку інформаційних технологій та зростання складності програмних систем, тестування стає критично важливим елементом життєвого циклу розробки програмного забезпечення.

Тестування програмного забезпечення можна визначити як процес дослідження програмного продукту з метою отримання інформації про його якість відносно контексту, в якому він призначений для використання. Цей процес включає планування, підготовку, виконання та аналіз тестів, а також документування результатів. Основною метою тестування є виявлення відхилень від очікуваної поведінки системи та забезпечення відповідності програмного продукту заданим вимогам та стандартам якості.

Сучасні тенденції в області тестування програмного забезпечення характеризуються переходом від традиційних підходів до Agile та DevOps методологій, де тестування інтегроване в усі етапи розробки. Це включає безперервну інтеграцію, безперервне тестування та автоматизацію тестових процесів. Зростання популярності мікросервісної архітектури, хмарних технологій та контейнеризації також впливає на підходи до тестування, вимагаючи нових методів та інструментів для перевірки коректності роботи складних розподілених систем.

Багатокомпонентні веб-додатки являють собою складні системи, що складаються з кількох взаємопов'язаних компонентів, кожен з яких виконує специфічні функції. До таких компонентів належать серверна частина (backend), що забезпечує обробку бізнес-логіки та роботу з даними; веб-інтерфейс (frontend), що забезпечує взаємодію з користувачем; та інтеграційні компоненти, що забезпечують взаємодію з зовнішніми системами та сервісами. Тестування таких систем вимагає комплексного підходу, що охоплює всі компоненти та їх взаємодію між собою.

Нормативно-правова база в області тестування програмного забезпечення включає міжнародні та національні стандарти, що визначають вимоги до якості програмних продуктів та процесів їх розробки. Основним міжнародним

стандартом є ISO/IEC 25010 "Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models", який визначає модель якості програмного продукту, що включає вісім основних характеристик: функціональна придатність, продуктивність, сумісність, зручність використання, надійність, безпека, супроводжуваність та переносимість. Цей стандарт служить основою для розробки стратегій тестування, оскільки кожна характеристика якості вимагає специфічних методів перевірки.

В Україні ці вимоги відображені в національних стандартах ДСТУ, зокрема ДСТУ ISO/IEC 25010:2015, що гармонізований з міжнародним стандартом. Крім того, нормативні документи Міністерства цифрової трансформації України та Міністерства освіти і науки України встановлюють вимоги до якості інформаційних систем, що використовуються в державному секторі та освітніх закладах. Ці документи підкреслюють важливість комплексного тестування як засобу забезпечення якості програмних продуктів.

Класифікація методів тестування базується на різних критеріях, включаючи рівень тестування, метод здійснення, ступінь автоматизації та тип перевіряємої характеристики. За рівнем тестування виділяють чотири основних рівні: unit-тестування, integration-тестування, system-тестування та acceptance-тестування. Unit-тестування (модульне тестування) спрямоване на перевірку окремих компонентів або функцій системи в ізоляції від інших компонентів. Це найнижчий рівень тестування, що дозволяє виявляти дефекти на ранніх етапах розробки та забезпечує коректність роботи окремих модулів.

Integration-тестування (інтеграційне тестування) перевіряє коректність взаємодії між різними компонентами системи. Цей рівень тестування є критично важливим для багатокомпонентних систем, оскільки навіть якщо кожен компонент окремо працює коректно, їх взаємодія може містити дефекти. Integration-тестування може проводитися як між модулями одного рівня (наприклад, між різними API endpoints), так і між компонентами різних рівнів (наприклад, між frontend та backend).

System-тестування (системне тестування) перевіряє систему в цілому як повністю інтегрований продукт для перевірки відповідності вимогам. Цей рівень тестування включає функціональне та нефункціональне тестування, включаючи тестування продуктивності, безпеки, надійності та інших характеристик якості. System-тестування проводиться в середовищі, що максимально наближене до реального експлуатаційного середовища.

Acceptance-тестування (приймальне тестування) перевіряє відповідність системи вимогам замовника та готовність до впровадження. Цей рівень тестування може проводитися як замовником, так і незалежною командою

тестування, і включає перевірку того, що система вирішує задачі користувачів та відповідає їхнім очікуванням.

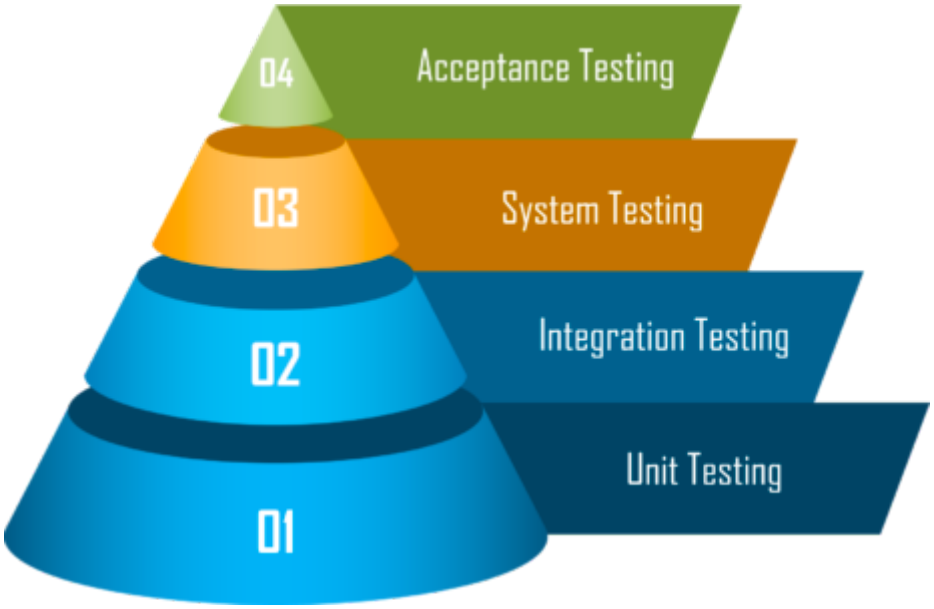


Рис. 1. Ієрархічна “Піраміда тестування”

За методом здійснення тестування виділяють тестування "чорної скриньки" (black-box testing), де внутрішня структура системи невідома, та тестування "білої скриньки" (white-box testing), де тестувальник має доступ до вихідного коду та внутрішньої структури системи. Тестування "чорної скриньки" фокусується на перевірці функціональності системи з точки зору користувача, тоді як тестування "білої скриньки" дозволяє перевіряти внутрішню логіку та покриття коду.

За ступенем автоматизації виділяють ручне тестування та автоматизоване тестування. Ручне тестування виконується тестувальником вручну без використання автоматизованих інструментів і є ефективним для дослідницького тестування, тестування користувацького досвіду та ситуацій, що важко автоматизувати. Автоматизоване тестування використовує спеціалізовані інструменти та скрипти для автоматичного виконання тестів, що дозволяє значно підвищити ефективність та скоротити час тестування, особливо для повторюваних сценаріїв.

Функціональна модель процесу тестування багатокомпонентної системи включає кілька основних етапів: планування тестування, розробку тестових сценаріїв, підготовку тестового середовища, виконання тестів, аналіз результатів та звітування. Планування тестування включає визначення стратегії тестування, вибір рівнів та типів тестів, визначення критеріїв успіху та розподіл ресурсів. Розробка тестових сценаріїв включає створення детальних описів тестів, включаючи вхідні дані, очікувані результати та кроки виконання.

Підготовка тестового середовища включає налаштування необхідної інфраструктури, включаючи тестові бази даних, тестові сервери, тестові дані та

конфігурацію системи. Для багатокомпонентних систем це може включати налаштування окремих середовищ для кожного компонента та забезпечення їх взаємодії. Виконання тестів включає фактичний запуск тестів, збір результатів та моніторинг виконання. Аналіз результатів включає інтерпретацію отриманих даних, виявлення дефектів та визначення їх причин. Звітування включає документування результатів тестування, включаючи статистику проходження тестів, виявлені дефекти та рекомендації щодо покращення.

До основних акторів(учасників процесу) належать тестувальник (QA інженер), розробник, автоматизована система тестування, тести різних рівнів, звіти про результати тестування. Тестувальник розробляє та виконує тести, аналізує результати та документує виявлені дефекти. Розробник забезпечує підтримку тестової інфраструктури, виправляє виявлені дефекти та бере участь у розробці тестових сценаріїв. Автоматизована система тестування виконує тести та генерує звіти, тести перевіряють різні аспекти системи, звіти надають інформацію про стан якості продукту.

Процес починається з аналізу вимог до системи та планування стратегії тестування, що включає визначення обсягу тестування, вибір методів та інструментів, а також розподіл ресурсів. Наступним етапом є розробка тестових сценаріїв, що включає створення детальних описів тестів для різних рівнів та компонентів системи.

Після розробки тестових сценаріїв проводиться налаштування тестового середовища, що включає підготовку тестових баз даних, конфігурацію тестових серверів, створення тестових даних та налаштування інтеграції між компонентами. Виконання тестів різних рівнів проводиться послідовно, починаючи з unit-тестів, далі integration-тестів, system-тестів та acceptance-тестів. Кожен рівень тестування має свої цілі та методи виконання.

Після виконання тестів проводиться аналіз результатів, що включає перевірку проходження тестів, аналіз виявлених дефектів та визначення їх причин. Якщо виявлено дефекти, вони передаються розробникам для виправлення, після чого проводиться повторне тестування. Процес є ітераційним і може повторюватися кілька разів до досягнення прийнятного рівня якості. Фінальним етапом є формування звіту про якість, що включає статистику тестування, виявлені дефекти, рекомендації та оцінку готовності системи до впровадження.

Специфіка тестування різних компонентів багатокомпонентної системи вимагає застосування різних підходів та інструментів. Тестування серверної частини (API) фокусується на перевірці коректності обробки запитів, валідації вхідних даних, обробки помилок, продуктивності та безпеки. Для API тестування використовуються інструменти, що дозволяють надсилати

HTTP-запити та перевіряти відповіді, включаючи статус-коди, тіло відповіді, заголовки та час відповіді. Важливим аспектом є тестування різних сценаріїв, включаючи позитивні та негативні тести, граничні значення та обробку виняткових ситуацій.

Тестування веб-інтерфейсу фокусується на перевірці коректності відображення елементів інтерфейсу, обробки користувацьких дій, навігації між сторінками, валідації форм та відповідності дизайну. Для веб-тестування використовуються інструменти, що дозволяють симулювати дії користувача в браузері, включаючи кліки, введення тексту, навігацію та перевірку результатів. Важливим аспектом є крос-браузерне тестування, що забезпечує коректну роботу в різних браузерах, а також тестування адаптивного дизайну для різних пристроїв.

Тестування інтеграційних компонентів, зокрема Telegram ботів, має свою специфіку, пов'язану з необхідністю симуляції роботи зовнішніх API та перевірки логіки обробки повідомлень. Тестування Telegram бота включає перевірку коректності обробки команд від користувачів, валідації вхідних даних, інтеграції з серверною частиною, обробки помилок та синхронізації даних між різними каналами доступу. Особливістю є необхідність тестування в реальному середовищі Telegram або використання моків для симуляції роботи Telegram API.

Особливістю тестування систем з месенджер-інтеграцією є необхідність перевірки коректної роботи в умовах асинхронної комунікації, обробки черг повідомлень, роботи з вебхуками та довгого опитування. Також важливим є тестування багатокористувацьких сценаріїв, де кілька користувачів одночасно взаємодіють з ботом, та перевірка ізоляції даних між різними користувачами. Додатковим аспектом є тестування оновлень та змін у функціональності бота, включаючи перевірку сумісності з різними версіями Telegram API та обробку змін в інтерфейсі месенджера.

Таким чином, предметне середовище тестування багатокомпонентних веб-додатків являє собою складну систему, що вимагає комплексного підходу, включаючи різні рівні та методи тестування, спеціалізовані інструменти та врахування специфіки кожного компонента. Сучасні тенденції в області тестування спрямовані на автоматизацію процесів, інтеграцію тестування в розробку та забезпечення високої якості програмних продуктів на всіх етапах життєвого циклу.

1.2. Огляд наявних аналогів

Сучасний ринок інструментів для тестування програмного забезпечення пропонує широкий спектр рішень, що розрізняються за функціональністю, вартістю, складністю впровадження та цільовою аудиторією. Класифікація цих інструментів базується на типі тестування, що вони підтримують, рівні автоматизації, технологічному стеці та сфері застосування. Для комплексного тестування багатокомпонентних веб-додатків необхідно використовувати набір інструментів, що охоплюють різні аспекти якості програмного продукту.

Інструменти для тестування можна класифікувати за наступними критеріями: за типом тестування (функціональне, нефункціональне), за рівнем тестування (unit, integration, system, acceptance), за методом автоматизації (ручні, автоматизовані), за технологічним стеком (для конкретних мов програмування та фреймворків), за моделлю ліцензування (відкриті, комерційні) та за сферою застосування (для веб, мобільних, настільних додатків). Така класифікація дозволяє системно підійти до вибору інструментарію для конкретного проєкту.

Інструменти для API тестування є критично важливими для перевірки серверної частини веб-додатків. Одним з найпопулярніших інструментів є Postman - платформа для розробки та тестування API, що надає зручний інтерфейс для створення та виконання HTTP-запитів, організації їх у колекції, автоматизації тестів та генерації документації. Postman підтримує різні типи автентифікації, зберігання змінних середовища та інтеграцію з CI/CD системами. Основними перевагами Postman є зручність використання, широкі можливості для автоматизації та велика спільнота користувачів. Однак, для складних сценаріїв автоматизації може знадобитися додаткове знання та навички програмування на JavaScript.

Swagger (OpenAPI) є іншим популярним інструментом для API тестування, що базується на специфікації OpenAPI. Swagger дозволяє автоматично генерувати документацію API на основі коду, створювати інтерактивні інтерфейси для тестування ендпоінтів та валідувати відповіді відповідно до специфікації. Інтеграція Swagger з фреймворками такими як FastAPI дозволяє автоматично генерувати документацію та тести на основі анотацій в коді. Перевагами Swagger є тісна інтеграція з сучасними фреймворками, автоматична генерація документації та підтримка стандартів. Недоліком є обмежена гнучкість для складних сценаріїв тестування.

Для програмної автоматизації API тестування широко використовуються бібліотеки мов програмування, зокрема pytest для Python. Pytest є потужним фреймворком для тестування, що підтримує написання тестів різних рівнів, включаючи unit, integration та functional тести. У комбінації з бібліотекою requests для виконання HTTP-запитів, pytest дозволяє створювати гнучкі та

масштабовані тести для API. Перевагами такого підходу є повна гнучкість програмування, можливість інтеграції з іншими Python бібліотеками, підтримка паралельного виконання тестів та генерація детальних звітів. Недоліком є необхідність програмування та більша складність налаштування порівняно з графічними інструментами.

Інструменти для тестування веб-інтерфейсів забезпечують перевірку коректності відображення та функціонування користувацького інтерфейсу в браузері. Selenium є одним з найстаріших та найпопулярніших інструментів для автоматизації веб-тестування. Selenium підтримує різні браузери через WebDriver, дозволяє симулювати дії користувача (кліки, введення тексту, навігація) та перевіряти стан елементів сторінки. Selenium підтримує різні мови програмування (Java, Python, JavaScript, C#) та має велику екосистему інструментів. Перевагами Selenium є широка підтримка браузерів, гнучкість та велика спільнота відповідна за розмірами база знань та відповідей. Недоліками є складність налаштування, повільність виконання тестів та проблеми зі стабільністю тестів.

Playwright є сучаснішою альтернативою Selenium, розробленою Microsoft. Playwright підтримує всі основні браузери (Chrome, Firefox, Safari, Edge), забезпечує швидше виконання тестів завдяки використанню протоколу DevTools та має кращу стабільність тестів. Playwright підтримує JavaScript, TypeScript, Python, .NET та Java, надає зручні API для очікування елементів, роботи з мережевими запитами та створення скріншотів. Перевагами Playwright є швидкість, стабільність, сучасний API та вбудовані можливості для тестування мобільних браузерів. Недоліком є менша спільнота порівняно з Selenium та відносно новий інструмент.

Cypress є ще одним популярним інструментом для тестування веб-інтерфейсів, що позиціонується як "швидкий, легкий та надійний" інструмент для end-to-end тестування. Cypress працює безпосередньо в браузері, що забезпечує швидке виконання тестів та легке налагодження. Cypress має зручний інтерфейс для перегляду виконання тестів в реальному часі, підтримує автоматичне очікування елементів та має вбудовані можливості для mock-ування мережевих запитів. Перевагами Cypress є швидкість, зручність налагодження та простота використання. Недоліками є обмежена підтримка браузерів (переважно Chrome-подібні) та специфічний API, що відрізняється від стандартних підходів.

Puppeteer є бібліотекою від Google для керування браузером Chrome/Chromium через DevTools Protocol. Puppeteer дозволяє автоматизувати різні дії в браузері, включаючи створення скріншотів, генерацію PDF, виконання тестів та скрапінг даних. Puppeteer часто використовується в комбінації з іншими фреймворками для тестування, такими як Jest або Playwright.

Перевагами Puppeteer є швидкість, тісна інтеграція з Chrome та зручний API. Недоліком є обмежена підтримка інших браузерів та необхідність додаткових бібліотек для повноцінного тестування.

Інструменти для performance тестування дозволяють оцінити продуктивність системи під навантаженням. K6 є сучасним інструментом для load testing, що використовує JavaScript для написання тестів скриптів. K6 підтримує різні типи навантажень, включаючи stress, spike, soak та surge тестування, забезпечує детальну звітність та інтеграцію з CI/CD системами. Перевагами K6 є простота використання, сучасний підхід до написання тестів та хороша інтеграція з хмарними сервісами. Недоліком є обмежена підтримка розподілених тестів без додаткових інструментів.

Apache JMeter є класичним інструментом для performance та load testing, що використовується вже багато років. JMeter підтримує різні протоколи (HTTP, JDBC, JMS, FTP), має графічний інтерфейс для створення тестових планів та забезпечує детальну звітність. JMeter написаний на Java, що дозволяє використовувати його на різних платформах та розширювати функціональність через плагіни. Перевагами JMeter є гнучкість, широкі можливості та велика спільнота. Недоліками є складність використання, повільність роботи графічного інтерфейсу та застарілий підхід до написання тестів.

Gatling є інструментом для load testing, що використовує Scala для написання тестових сценаріїв. Gatling позиціонується як інструмент для високопродуктивного тестування, що дозволяє симулювати тисячі одночасних користувачів з одного вузла. Gatling має зручний DSL для опису тестових сценаріїв, забезпечує детальну звітність з графіками та інтегрується з CI/CD системами. Перевагами Gatling є висока продуктивність, сучасний підхід та детальна звітність. Недоліком є необхідність вивчення Scala та менша спільнота порівняно з JMeter.

Locust є інструментом для load testing на базі Python, що використовує зрозумілий синтаксис для написання тестів. Locust дозволяє легко масштабувати тестування на кілька вузлів, забезпечує веб-інтерфейс для моніторингу виконання тестів в реальному часі та підтримує різні протоколи через HTTP бібліотеки. Перевагами Locust є простота використання для Python розробників, легке масштабування та зручний веб-інтерфейс. Недоліком є менша продуктивність порівняно з Gatling для дуже великих навантажень.

Інструменти для security тестування дозволяють виявляти вразливості в програмному забезпеченні. OWASP ZAP (Zed Attack Proxy) є безкоштовним інструментом для безпеки веб-додатків, що розробляється спільнотою OWASP. ZAP дозволяє автоматично сканувати веб-додатки на наявність вразливостей, інтерактивно тестувати ендпоінти та аналізувати трафік. ZAP підтримує різні

типи сканувань, включаючи SQL Injection, XSS, CSRF та інші атак згідно з OWASP Top 10. Перевагами ZAP є безкоштовність, широкі можливості та активна спільнота. Недоліком є складність використання для новачків та необхідність налаштування для коректної роботи.

Burp Suite є комерційним інструментом для security тестування веб-додатків, що широко використовується професіоналами з безпеки. Burp Suite включає інтерактивний проксі для перехоплення та модифікації HTTP-запитів, автоматичний сканер вразливостей, інструменти для fuzz-тестування та аналізу трафіку. Burp Suite має безкоштовну версію з обмеженим функціоналом та платну версію з розширеними можливостями. Перевагами Burp Suite є потужний функціонал, зручний інтерфейс та широкі можливості для ручного тестування. Недоліком є висока вартість платної версії та складність освоєння.

SonarQube є платформою для безперервної перевірки якості коду, що включає виявлення вразливостей, code smells та багів. SonarQube аналізує вихідний код на предмет відповідності стандартам якості, виявляє потенційні проблеми з безпекою та надає рекомендації щодо покращення. SonarQube підтримує різні мови програмування та інтегрується з CI/CD системами. Перевагами SonarQube є комплексний підхід до якості коду, детальна звітність та інтеграція з процесом розробки. Недоліком є складність налаштування та необхідність серверної інфраструктури.

Інструменти для тестування доступності дозволяють перевірити відповідність веб-інтерфейсів стандартам доступності, таким як WCAG 2.1. axe DevTools є популярним інструментом для тестування доступності, що доступний як розширення для браузерів та як бібліотека для автоматизації. axe DevTools автоматично сканує веб-сторінки на наявність проблем з доступністю, надає детальні звіти з рекомендаціями щодо виправлення та підтримує різні рівні відповідності WCAG. Перевагами axe DevTools є точність, простота використання та підтримка стандартів. Недоліком є обмежена можливість перевірки контексту використання.

Lighthouse є інструментом від Google для аудиту веб-сторінок, що включає перевірку доступності, продуктивності, SEO та best practices. Lighthouse доступний як розширення для Chrome, як інструмент командного рядка та інтегрується в Chrome DevTools. Lighthouse надає детальні звіти з балами за різними категоріями та конкретними рекомендаціями щодо покращення. Перевагами Lighthouse є безкоштовність, комплексний підхід та інтеграція з Chrome. Недоліком є обмежена глибина аналізу порівняно зі спеціалізованими інструментами.

WAVE (Web Accessibility Evaluation Tool) є безкоштовним інструментом для оцінки доступності веб-сторінок, що розробляється WebAIM. WAVE доступний як онлайн-сервіс та як розширення для браузерів, дозволяє візуально відображати проблеми з доступністю безпосередньо на сторінці та надає детальні рекомендації. Перевагами WAVE є простота використання та зручне візуальне відображення проблем. Недоліком є обмежена можливість автоматизації порівняно з іншими інструментами.

Порівняльний аналіз інструментів за критеріями функціональності, вартості та складності впровадження демонструє, що немає універсального рішення, що підходить для всіх випадків. Відкриті рішення, такі як pytest, Playwright, K6, OWASP ZAP, axe DevTools, пропонують гнучкість та відсутність ліцензійних витрат, але часто вимагають більше часу на налаштування та навчання. Комерційні рішення, такі як платна версія Burp Suite, пропонують кращу підтримку та розширені можливості, але мають значну вартість.

Для досліджуваної системи управління розкладом оптимальним набором інструментів є комбінація відкритих рішень, що забезпечують комплексне тестування при мінімальних витратах. Для API тестування доцільно використовувати pytest з бібліотекою requests, що забезпечує гнучкість та інтеграцію з існуючою Python-інфраструктурою. Для тестування веб-інтерфейсу Playwright є оптимальним вибором завдяки швидкості, стабільності та підтримці різних браузерів. Для performance тестування K6 забезпечує простоту використання та хорошу інтеграцію з CI/CD. Для security тестування OWASP ZAP надає безкоштовне та потужне рішення для виявлення вразливостей. Для тестування доступності axe DevTools та Lighthouse забезпечують комплексну перевірку відповідності стандартам WCAG.

Таким чином, огляд наявних аналогів демонструє широкий вибір інструментів для тестування програмного забезпечення, кожен з яких має свої переваги та недоліки. Вибір оптимального набору інструментів залежить від конкретних вимог проєкту, технологічного стеку, бюджету та експертизи команди. Для досліджуваної системи обраний набір відкритих інструментів забезпечує комплексне тестування при мінімальних витратах та максимальній гнучкості.

1.3. Постановка задачі

Метою даної дипломної роботи є розробка, документація та впровадження комплексної системи тестування для багатокомпонентного веб-додатку управління розкладом та університетською інформацією, що забезпечує високий рівень надійності, безпеки та доступності програмного продукту. Досягнення цієї мети вимагає вирішення низки взаємопов'язаних завдань, що охоплюють

аналіз предметної області, проектування тестової інфраструктури, вибір оптимального набору інструментів та практичне впровадження розроблених рішень.

Конкретні завдання дослідження включають: аналіз предметної області тестування багатокомпонентних веб-додатків, включаючи вивчення сучасних підходів, методологій та інструментальних засобів; проектування архітектури комплексної системи тестування для серверної частини на базі FastAPI, веб-інтерфейсу на Vue.js та інтеграційного компонента Telegram бота; вибір інструментарію для автоматизованого тестування різних рівнів, включаючи unit, integration, E2E, performance та security тести; розробку методики тестування доступності та валідації веб-інтерфейсів відповідно до стандартів WCAG; впровадження розробленої системи тестування в практику та проведення комплексного тестування досліджуваної системи; аналіз отриманих результатів та формування рекомендацій щодо покращення якості програмного продукту.

Практична цінність роботи полягає у значному підвищенні якості програмного продукту через впровадження комплексної системи тестування. Розроблена система тестування дозволяє виявляти дефекти на різних етапах розробки, що зменшує витрати на їх пошук і виправлення в пізні етапи розробки та підвищує стабільність роботи системи. Автоматизація тестових процесів дозволяє прискорити розробку нових функцій та забезпечити швидкий зворотний зв'язок про якість коду. Впровадження performance та security тестування дозволяє виявити потенційні проблеми з продуктивністю та безпекою до впровадження системи в експлуатацію. Перевірка доступності веб-інтерфейсу забезпечує відповідність стандартам WCAG та розширює аудиторію користувачів, включаючи людей з обмеженими можливостями. Розроблена методологія може бути адаптована для інших веб-додатків з подібною архітектурою, що підтверджує її універсальність та практичну цінність для галузі розробки програмного забезпечення.

Очікувані результати роботи включають: розроблену архітектуру комплексної системи тестування для багатокомпонентного веб-додатку; впроваджений набір автоматизованих тестів різних рівнів, що охоплюють серверну частину та інтеграційний компонент; систему звітності про результати тестування з детальною статистикою та метриками якості; рекомендації щодо покращення архітектури та коду на основі результатів тестування; документацію методики тестування, що може бути використана для інших проєктів; оцінку ефективності впровадженої системи тестування через показники якості, такі як кількість виявлених дефектів, час виконання тестів, покриття коду та продуктивність системи.

Методи дослідження, що будуть використані в роботі, включають: аналіз літературних джерел та нормативної документації з питань тестування програмного забезпечення; порівняльний аналіз існуючих інструментів та методологій тестування; експериментальне впровадження розроблених рішень з використанням обраних інструментів тестування; емпіричний аналіз результатів тестування та оцінка ефективності запропонованих підходів; систематизація отриманих результатів та формування узагальнених висновків та рекомендацій. Комбінація теоретичних та практичних методів дозволяє забезпечити наукову обґрунтованість та практичну значущість отриманих результатів.

Таким чином, постановка задачі визначає чітку мету та конкретні завдання дослідження, об'єкт та предмет дослідження та практичну цінність роботи, очікувані результати та методи їх досягнення. Це забезпечує системний підхід до виконання дипломної роботи та дозволяє сфокусуватися на ключових аспектах комплексного тестування багатокомпонентного веб-додатку.

Висновки до розділу І

У першому розділі було проведено всебічне дослідження теоретичних та практичних засад тестування сучасних багатокомпонентних веб-систем, що дозволило сформувати цілісну картину предметного середовища. Аналіз показав, що в умовах стрімкого розвитку Agile-методологій та мікросервісної архітектури тестування трансформувалося з фінального етапу розробки на безперервний процес, інтегрований у кожен крок життєвого циклу програмного забезпечення. Особлива складність досліджуваного об'єкта, що поєднує сервер на FastAPI, інтерфейс на Vue.js та інтеграцію з Telegram, диктує необхідність застосування комплексного підходу, де кожен рівень — від модульних перевірок до системного аналізу — виконує свою специфічну роль у забезпеченні якості.
Важливим аспектом роботи стало вивчення нормативно-правової бази, зокрема стандарту ISO/IEC 25010, який слугує методологічним орієнтиром для оцінки функціональності, безпеки та зручності системи. Огляд широкого спектра сучасних інструментів, таких як Postman, Selenium, Playwright та JMeter, дозволив дійти висновку, що для даного проєкту найбільш виправданим є використання стеку відкритих технологій. Поєднання pytest для серверної логіки, Playwright для браузерної автоматизації та інструментів типу K6 для навантажувального тестування створює гнучку та економічно ефективну інфраструктуру, спроможну забезпечити високу стабільність продукту.

Окрему увагу було приділено питанню інклюзивності, де тестування доступності згідно зі стандартами WCAG визначено як критично важливий елемент сучасної розробки, що дозволяє зробити університетську інформаційну систему відкритою для всіх категорій користувачів. Підсумовуючи результати

розділу, можна стверджувати, що поставлені завдання з аналізу ринку та проєктування архітектури тестування виконані в повному обсязі. Це створює надійний теоретичний фундамент для подальшої практичної реалізації системи тестування, спрямованої на мінімізацію ризиків, оптимізацію витрат на розробку та гарантування безпеки і надійності фінального програмного рішення.

РОЗДІЛ ІІ. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Аналіз предметної області(вхідні та вихідні дані)

Аналіз предметної області тестування системи управління розкладом вимагає виділення основних об'єктів дослідження з точки зору їх тестування. Основними об'єктами тестування є компоненти системи: серверна частина (API), веб-інтерфейс та Telegram бот. Кожен компонент має специфічні вимоги до вхідних та вихідних даних для тестування, що визначає стратегію підготовки тестових даних та вибір методів тестування.

2.1.1. Серверна частина

Для тестування серверної частини (API) вхідними даними є HTTP-запити з різними методами (GET, POST, PUT, DELETE), що містять параметри шляху (path parameters), параметри запиту (query parameters), тіло запиту (request body) та заголовки (headers). Важливим аспектом є підготовка тестових даних для API, що включає створення тестових користувачів з різними ролями (студент, викладач, адміністратор, менеджер розкладу), тестових груп, викладачів, аудиторій, дисциплін та розкладів. Ці дані мають бути реалістичними, але не містити конфіденційної інформації. Для тестування автентифікації та авторизації використовуються тестові токени JWT, що генеруються для різних ролей користувачів.

Вихідними даними для тестування API є HTTP-відповіді зі статус-кодами, тілом відповіді (response body) та заголовками. Тестування API включає перевірку статус-кодів (200 для успішних операцій, 400 для помилок валідації, 401 для помилок автентифікації, 404 для відсутніх ресурсів, 500 для внутрішніх помилок), структури тіла відповіді (наявність очікуваних полів, типи даних, валідність значень) та заголовків (Content-Type, Authorization). Для negative тестування використовуються некоректні вхідні дані, що перевіряють обробку помилок валідації та граничні випадки.

2.1.2. Веб-інтерфейс

Для тестування веб-інтерфейсу вхідними даними є дії користувача (кліки, введення тексту, навігація), конфігурація браузера (тип браузера, розмір вікна, user agent) та стан додатку (залогінений користувач, поточна сторінка, дані в формах). Підготовка тестових даних для веб-інтерфейсу включає налаштування тестового середовища, створення тестових користувачів, підготовку тестових даних в базі даних та конфігурування API для тестування. Для E2E тестування

важливо забезпечити ізоляцію тестів, щоб кожен тест працював з чистим станом системи.

Вихідними даними для тестування веб-інтерфейсу є стан елементів сторінки (видимість, активність, текстове содержим), URL поточної сторінки, повідомлення про помилки або успішні операції, а також поведінка додатку при виконанні дій. Тестування веб-інтерфейсу включає перевірку коректності відображення даних, працюючості форм валідації, навігації між сторінками та обробки помилок. Для cross-browser тестування використовуються різні браузери з різними версіями, що вимагає адаптації тестових даних та очікуваних результатів. Також тестування системи доступності. Ручне тестування сторінки та документація очікуваних та фактичних значень, змін елементів, дій, навігації, картинки).

2.1.3. Зовнішній сервіс Telegram бот

Для тестування Telegram бота вхідними даними є повідомлення від користувачів (текстові команди, callback-кнопки, inline-кнопки), метадані повідомлень (ID користувача, ID чату, timestamp) та стан бота (поточний контекст розмови, дані сесії). Підготовка тестових даних для бота включає створення тестових користувачів з різними ролями, підготовку тестових повідомлень з різними командами та налаштування моків для API сервера, щоб тестувати логіку бота без залежності від реального API.

Вихідними даними для тестування Telegram бота є повідомлення, що відправляються користувачам (текст, inline-кнопки, callback-кнопки), зміни стану бота (оновлення контексту розмови, збереження даних сесії) та виклики API сервера. Тестування бота включає перевірку коректності обробки команд, валідації вхідних даних, формування відповідей та обробки помилок. Для тестування інтеграції з API використовуються моки, що імітують відповіді API сервера, що дозволяє тестувати різні сценарії (успішні відповіді, помилки, таймаути).

2.1.4. Вхідні та вихідні дані

Існуючі обмеження на вхідні та вихідні дані для тестування включають правила валідації, що мають бути враховані при створенні тестових даних. Для API тестування email-адреси мають відповідати формату та бути унікальними, паролі мають мінімальну довжину та складність, ID об'єктів мають бути валідними та відповідати існуючим записам. Для веб-інтерфейсу форми мають правила валідації, що перевіряються в тестах. Для Telegram бота команди мають

відповідати визначеному формату, повідомлення мають обмеження за довжиною (4096 символів), кнопки мають обмеження за кількістю та розміром.

Обмеження на вихідні дані включають правила форматування відповідей та обмеження обсягу даних. API відповіді мають обмеження за кількістю записів (зазвичай 100 на сторінку) для запобігання перевантаження сервера, що враховується в тестах. Повідомлення Telegram бота мають обмеження за довжиною, що вимагає розбиття довгих повідомлень на частини або скорочення тексту в тестах. Веб-інтерфейс має обмеження за кількістю елементів на сторінці, що перевіряється в тестах продуктивності.

2.1.5. Специфіка та генерація тестових даних

Специфіка тестових даних включає використання різних підходів для генерації та управління даними в тестах. Для unit-тестів використовуються моки (mocks) та стаби (stubs), що імітують зовнішні залежності та дозволяють тестувати компоненти в ізоляції. Наприклад, для тестування сервісу автентифікації мокається база даних, що повертає predefined користувачів. Для integration-тестів використовується реальна тестова база даних, що ізольована від продакшн, та створюються реальні HTTP-запити до API. Для E2E-тестів створюється повне тестове середовище, що включає базу даних, API сервер, веб-інтерфейс та Telegram бота.

Фікстури (fixtures) використовуються для створення повторюваних тестових даних. Наприклад, фікстура "test_user" створює користувача з predefined параметрами, що використовується в декількох тестах. Фікстури забезпечують ізоляцію тестів та дозволяють легко керувати тестовими даними. Для pytest використовуються conftest.py файли з визначенням фікстур, що автоматично створюються перед виконанням тестів та очищаються після завершення.

Генерація тестових даних використовує різні підходи: статичні дані (predefined значення), динамічна генерація (випадкові значення в межах правил валідації) та фабрики даних (factory pattern для створення складних об'єктів). Наприклад, для генерації тестових email-адрес використовуються випадкові рядки з доменом @test.example.com, для генерації дат використовуються випадкові дати в певному діапазоні, для створення складних об'єктів (розклад) використовуються фабрики, що автоматично створюють пов'язані об'єкти (групу, викладача, аудиторію, дисципліну).

2.1.6. Потоки та ізоляція

Потоки тестових даних між компонентами під час тестування мають специфічну структуру. Для unit-тестів потік даних є локальним в межах тесту: вхідні дані передаються безпосередньо в функцію, вихідні дані повертаються та перевіряються. Для integration-тестів потік даних включає кілька компонентів: тест надсилає HTTP-запит до API, API обробляє запит, звертається до бази даних, повертає відповідь, тест перевіряє відповідь. Для E2E-тестів потік даних включає всі компоненти системи: користувач взаємодіє з веб-інтерфейсом, веб-інтерфейс надсилає запити до API, API звертається до бази даних, результати повертаються через веб-інтерфейс користувачу.

Ізоляція тестових даних є критично важливою для забезпечення стабільності тестів. Кожен тест повинен працювати з незалежним набором даних, щоб результат одного тесту не впливав на інші. Це досягається через використання транзакцій бази даних, що відкотюються після завершення тесту, створення унікальних ідентифікаторів для кожного тесту, очищення тестових даних після виконання тесту та використання окремих тестових баз даних для різних наборів тестів.

2.1.7. Типи тестування за специфікою даних

Специфіка даних для різних типів тестування включає різні підходи. Для positive тестування використовуються коректні дані, що відповідають всім правилам валідації та очікуваним сценаріям використання. Для negative тестування використовуються некоректні дані, що порушують правила валідації (некоректні формати, відсутні обов'язкові поля, значення поза діапазоном). Для граничного тестування (edge case testing) використовуються дані на межах допустимих значень (мінімальні та максимальні значення, порожні рядки, нульові значення). Для security тестування використовуються дані, що імітують атаки (SQL injection, XSS, CSRF).

Таким чином, аналіз предметної області тестування системи управління розкладом дозволив виділити основні об'єкти тестування, описати вхідні та вихідні дані для різних компонентів системи, визначити обмеження на дані, описати специфіку тестових даних та потоки даних між компонентами під час тестування. Цей аналіз є основою для проектування системи тестування та розробки математичного та алгоритмічного забезпечення.

2.2. Проектування системи

2.2.1. Загальна архітектура системи тестування

Концептуальне проектування системи тестування базується на архітектурі досліджуваної багатокомпонентної системи управління розкладом та вимогах до комплексного тестування. Архітектура системи тестування включає три основні компоненти: тестову інфраструктуру для API, тестову інфраструктуру для веб-інтерфейсу та тестову інфраструктуру для Telegram бота. Кожен компонент має специфічну архітектуру, що враховує особливості тестування відповідної частини системи, але всі компоненти інтегровані в єдину систему тестування з спільною базою тестових даних, системою звітності та інтеграцією з CI/CD процесами.

Архітектура тестування API базується на багаторівневій моделі, що включає unit-тестування, integration-тестування та E2E-тестування. Unit-тестування фокусується на тестуванні окремих функцій та класів серверної частини в ізоляції від зовнішніх залежностей. Для цього використовується архітектура з використанням моків (mocks) та стабів (stubs), що імітують базу даних, зовнішні API та інші залежності. Integration-тестування перевіряє взаємодію між компонентами серверної частини, включаючи CRUD операції з базою даних, валідацію схем, автентифікацію та авторизацію. E2E-тестування API перевіряє повні сценарії використання, що імітують реальні запити від клієнтських додатків.

2.2.2. Архітектура тестування API та UML - структура

UML-діаграма класів тестової інфраструктури(див. Додаток А) для API включає основні класи та їх взаємозв'язки. Клас TestClient забезпечує виконання HTTP-запитів до API та перевірку відповідей. Клас TestFixture відповідає за створення та управління тестовими даними, включаючи створення тестових користувачів, груп, викладачів, аудиторій та розкладів. Клас DatabaseMock забезпечує імітацію бази даних для unit-тестів, дозволяючи тестувати бізнес-логіку без реальної бази даних. Клас APITest є базовим класом для всіх API тестів, що забезпечує спільну функціональність, включаючи налаштування тестового середовища, автентифікацію та очищення після тестів.

2.2.3. Структура тестової бази даних(ER-модель)

ER-діаграма тестової бази даних відображає структуру даних, що використовуються для integration та E2E тестування. Тестова база даних має таку саму структуру, як і продакшн база даних, але ізольована від неї та

заповнена тестовими даними. Основними таблицями є users (користувачі з різними ролями), roles (ролі користувачів: student, teacher, admin), groups (навчальні групи), teachers (викладачі), classrooms (аудиторії), subjects (дисципліни), schedules (розклади занять), assignments (завдання) та attendance (відвідуваність). Зв'язки між таблицями реалізовані через зовнішні ключі, що забезпечують цілісність даних. Для тестування також використовуються додаткові таблиці для ведення історії змін та аудиту, що дозволяють перевіряти коректність операцій оновлення та видалення.

Схема тестової бази даних включає індекси для оптимізації запитів, що важливо для performance тестування. Таблиці мають тригери для автоматичного оновлення timestamps та валідації даних. Для забезпечення ізоляції тестів кожен тест може використовувати окрему транзакцію, що відкотюється після завершення тесту, або окрему схему в базі даних. Це дозволяє виконувати тести паралельно без взаємного впливу. Тестова база даних також включає процедури для швидкого очищення та заповнення тестовими даними, що прискорює виконання тестів.

2.2.4. Тестування веб-інтерфейсу

Архітектура тестування веб-інтерфейсу базується на використанні фреймворку Playwright для end-to-end тестування. Тестова інфраструктура включає клас PageObject, що інкапсулює взаємодію з елементами веб-інтерфейсу, клас TestHelper для допоміжних функцій (очікування елементів, скріншоти, логування) та базовий клас WebTest для всіх веб-тестів. Page Object Pattern дозволяє ізолювати логіку тестів від деталей реалізації інтерфейсу, що робить тести більш стабільними при змінах дизайну.

Інтеграційні зв'язки між компонентами системи тестування забезпечують комплексний підхід до тестування. Тестова інфраструктура для API, веб-інтерфейсу та бота використовує спільну тестову базу даних, що забезпечує узгодженість тестових даних між різними типами тестів. Наприклад, користувач, створений в API тестах, може використовуватися в веб-тестах для перевірки автентифікації. Система звітності об'єднує результати всіх типів тестів в єдиний звіт з метриками якості, включаючи покриття коду, кількість пройдених/провалених тестів, час виконання та виявлені дефекти.

2.2.5. Інтеграція та CI/CD процеси

CI/CD інтеграція системи тестування включає автоматичний запуск тестів при кожному коміті в репозиторій, паралельне виконання тестів для скорочення часу, автоматичну генерацію звітів та сповіщень про результати. Тести поділені

на швидкі (unit тести), що запускаються при кожному коміті, та повільні (integration, E2E, performance тести), що запускаються nightly або при створенні pull request. Це забезпечує баланс між швидкістю зворотного зв'язку та глибиною тестування.

Архітектура управління тестовими даними включає фабрики (factories) для створення тестових об'єктів, фікстури (fixtures) для повторюваних наборів даних та систему очищення даних після тестів. Фабрики використовують шаблон Factory Method для створення складних об'єктів з валідними даними, що спрощує створення тестових сценаріїв. Фікстури забезпечують ізоляцію тестів через створення унікальних даних для кожного тесту. Система очищення включає автоматичне видалення створених даних після завершення тесту або використання транзакцій, що відкотюються.

Архітектура моніторингу та логування тестів включає збір метрик виконання тестів, логування операцій в базі даних, HTTP-запитів та відповідей, а також скріншотів при невдачах тестів. Це дозволяє швидко діагностувати причини невдач тестів та аналізувати продуктивність системи. Логи зберігаються в структурованому форматі (JSON) для подальшого аналізу та візуалізації. Метрики включають час виконання кожного тесту, пам'ять, що використовується, та кількість виконаних операцій з базою даних.

2.2.6. Підсумок проектування

Таким чином, проектування системи тестування включає архітектуру для комплексного тестування всіх компонентів досліджуваної системи, ER-діаграму тестової бази даних, UML-діаграми класів для тестової інфраструктури, інтеграційні зв'язки між компонентами та інтеграцію з CI/CD процесами (див. Рис. 2). Розроблена архітектура забезпечує масштабованість, ізоляцію тестів, швидке виконання та детальну звітність, що є критично важливим для ефективного тестування багатокомпонентної системи.

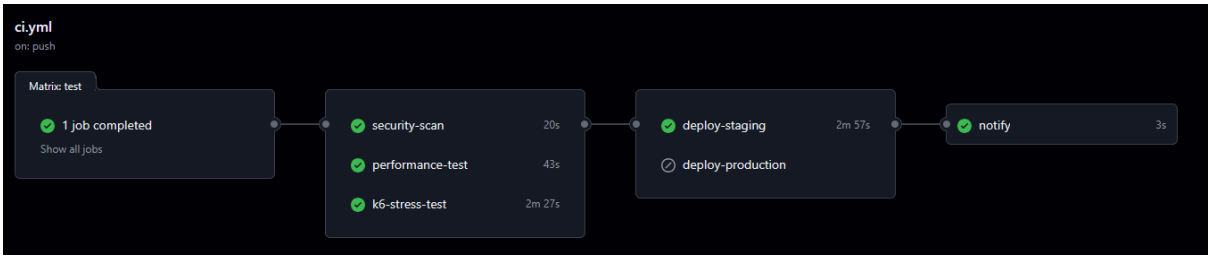


Рис. 2. CI/CD pipeline.

2.3. Математичне та алгоритмічне забезпечення

Математичне та алгоритмічне забезпечення системи тестування включає математичні моделі для оцінки ефективності тестування, алгоритми генерації

тестових даних, моделі оцінки продуктивності та алгоритми обробки результатів тестування. Ці моделі та алгоритми забезпечують наукову обґрунтованість підходів до тестування та дозволяють кількісно оцінювати якість та ефективність тестової системи.

2.3.1. Математичні моделі оцінки ефективності

Математичні моделі для тестування базуються на теорії ймовірностей та статистиці. Однією з основних метрик тестування є покриття коду (code coverage), що кількісно оцінює частину коду, яка була виконана під час тестування. Лінійне покриття (line coverage) розраховується як відношення кількості виконаних рядків коду до загальної кількості рядків коду:

$$Line\ Coverage = (Executed\ Lines / Total\ Lines) \times 100\%$$

Формула розрахунку % виконаних рядків коду.

Гілкове покриття (branch coverage) розраховується як відношення кількості виконаних гілок умовних переходів до загальної кількості гілок:

$$Branch\ Coverage = (Executed\ Branches / Total\ Branches) \times 100\%$$

Формула розрахунку відсотка виконаних умовних переходів

Покриття функцій (function coverage) розраховується як відношення кількості викликаних функцій до загальної кількості функцій:

$$Function\ Coverage = (Called\ Functions / Total\ Functions) \times 100\%$$

Формула розрахунку відсотка викликаних функцій

Ці метрики дозволяють оцінити, наскільки повно тести покривають код та ідентифікувати непокриті ділянки, що потребують додаткового тестування.

Математична модель ймовірності виявлення дефектів базується на теорії ймовірності та залежить від кількості тестів, складності системи та якості тестів. Імовірність того, що дефект буде виявлений після виконання n тестів, можна апроксимувати формулою:

$$P(defect\ detected) = 1 - (1 - p)^n$$

Формула розрахунку впевненості у відсутності дефектів

де p - ймовірність того, що один тест виявить конкретний дефект. Ця модель дозволяє оцінити необхідну кількість тестів для досягнення бажаного рівня впевненості у відсутності дефектів.

2.3.2. Алгоритми генерації тестових даних використовуються для створення валідних та різноманітних тестових даних.

Алгоритм генерації рядків з урахуванням правил валідації включає наступні кроки:

1) визначення правил валідації (довжина, дозволені символи, формат);

2) генерація випадкової довжини в межах допустимого діапазону;
3) послідовна генерація символів з дозволеного набору;
4) перевірка відповідності сформатованим правилам (наприклад, формат email);
5) повторення генерації при невідповідності правилам. Для генерації email-адрес використовується алгоритм, що генерує випадкову локальну частину та додає тестовий домен (наприклад, @test.example.com).

Алгоритм генерації дат для тестування включає:

1) визначення діапазону дат (мінімальна та максимальна дата);
2) генерація випадкового числа днів від початкової дати;
3) обчислення кінцевої дати;
4) форматування дати відповідно до ISO 8601 (YYYY-MM-DD);
5) перевірка на валідність (високосні роки, кількість днів у місяці). Для граничного тестування генеруються дати на межах діапазонів (наприклад, 1900-01-01, 2099-12-31) та спеціальні дати (29 лютого для високосних років).

Алгоритм генерації цілих чисел з урахуванням обмежень включає:

1) визначення мінімального та максимального значення;
2) генерація випадкового числа в цьому діапазоні;
3) для граничного тестування використання мінімального, максимального та нульового значень;
4) для negative тестування використання значень поза діапазоном. Для генерації ID об'єктів використовується алгоритм, що гарантує унікальність шляхом використання лічильника або генерації випадкових чисел з величезним діапазоном.

Алгоритми валідації вихідних даних використовуються в тестах для перевірки коректності обробки даних системою. Алгоритм валідації email включає:

1) перевірка наявності символу @;
2) перевірка наявність локальної частини до @;
3) перевірка наявності доменної частини після @;
4) перевірка наявність точки в доменній частині;
5) перевірка довжини локальної та доменної частини.

Алгоритм валідації пароля включає:

1) перевірка мінімальної довжини (зазвичай 8 символів);
2) перевірка наявності великих літер;
3) перевірка наявності малих літер;
4) перевірка наявності цифр;
5) перевірка наявності спеціальних символів.

2.3.3. Моделі оцінки продуктивності

Математичні моделі оцінки продуктивності використовуються для аналізу результатів performance тестування. Час відповіді API (response time) розраховується як різниця між часом отримання відповіді та часом відправки запиту:

$$Response\ Time = T\_response - T\_request$$

Формула розрахунку затримки між запитом та відповіддю

Середній час відповіді розраховується як середнє арифметичне часів відповідей для n запитів:

$$Average\ Response\ Time = (\Sigma\ Response\ Time\_i) / n$$

Формула розрахунку середнього часу обробки запитів

Медіана часу відповіді (P50) - це значення, що ділить відсортований список часів відповідей навпіл. Процентилі (P90, P95, P99) показують час відповіді, що не перевищується відповідно 90%, 95% та 99% запитів. Ці метрики дозволяють оцінити стабільність продуктивності та виявити аномально повільні запити.

Пропускна здатність (throughput) розраховується як кількість запитів, оброблених за одиницю часу:

$$Throughput = Number\ of\ Requests / Time\ Interval$$

Формула розрахунку кількості запитів за одиницю часу

Для load testing використовується модель розподілу навантаження, що описує, як запити розподіляються в часі. Найпоширенішими розподілами є: рівномірний розподіл (constant load), де запити надходять з постійною інтенсивністю; розподіл Пуассона, що моделює випадкові події з постійною середньою інтенсивністю; сплесковий розподіл (spike load), що імітує раптове збільшення навантаження.

Математична модель системи масового обслуговування (queueing theory) використовується для аналізу поведінки системи під навантаженням. Модель M/M/1 (Markovian arrival, Markovian service, 1 server) описує систему з одним сервером, де час між надходженням запитів та час обслуговування розподілені експоненційно. Середній час очікування в черзі розраховується як:

$$W\_q = \lambda / (\mu(\mu - \lambda))$$

Формула розрахунку середнього часу очікування в системі з одним сервером

де λ - інтенсивність надходження запитів, μ - інтенсивність обслуговування. Ця модель дозволяє оцінити, як система поводиться під навантаженням та виявити вузькі місця.

Алгоритми обробки помилок та виключень є критично важливими для надійності тестової системи.

2.3.4. Алгоритм безпеки та обробки помилок

Алгоритм обробки помилок API включає:

1) перехоплення виключення;

2) логування помилки з контекстом (час, тип помилки, стек виклику);

3) класифікація помилки (valiation error, authentication error, database error, internal error);

4) формування відповіді з відповідним статус-кодом та повідомленням про помилку;

5) очищення ресурсів (закриття з'єднань з базою даних, відкат транзакцій). Для retry-логіки використовується алгоритм експоненційного backoff, що збільшує інтервал між повторними спробами: 1s, 2s, 4s, 8s, 16s.

Алгоритм автентифікації та авторизації базується на використанні JSON Web Tokens (JWT). Процес генерації токена включає:

1) отримання credentials користувача (email, password);

2) валідація credentials;

3) отримання даних користувача з бази даних;

4) формування payload токена (user_id, role, expiration time);

5) підпис payload з використанням секретного ключа алгоритмом HMAC SHA256;

6) кодування в base64url та формування токена (header.payload.signature).

Процес валідації токена включає:

1) декодування tokenu;

2) перевірка підпису з використанням секретного ключа;

3) перевірка терміну дії токена;

4) отримання даних користувача з payload;

5) перевірка ролі та дозволів.

Алгоритм хешування паролів використовує bcrypt для безпечного зберігання паролів. Процес хешування включає:

1) генерація випадкової солі (salt);

2) додавання солі до пароля;

3) застосування функції хешування з ітераціями (cost factor);

4) збереження солі та хешу в базі даних.

Процес перевірки пароля включає:

1) отримання солі та хешу з бази даних;

2) хешування введеного пароля з тією ж сіллю;

3) порівняння отриманого хешу зі збереженим. Кількість ітерацій (cost factor) визначає стійкість до brute-force атак.

2.3.5. Управління результатами та звітність

Алгоритм генерації звітів про тестування включає:
1) збір результатів виконання тестів (статус, час виконання, повідомлення про помилки);
2) розрахунок метрик (кількість пройдених/провалених тестів, покриття коду, середній час виконання);
3) групування результатів за категоріями (unit, integration, E2E);
4) формування структурованого звіту (JSON, HTML);
5) генерація графіків та діаграм;
6) збереження звіту та відправка сповіщень. Для візуалізації результатів використовуються алгоритми побудови графіків (line chart для часу виконання тестів, pie chart для розподілу статусів, bar chart для покриття коду).
Алгоритм пріоритезації тестів дозволяє оптимізувати порядок виконання тестів для швидкого виявлення критичних дефектів. Алгоритм включає:
1) присвоєння пріоритету кожному тесту на основі критичності функціональності та історії виявлення дефектів;
2) сортування тестів за пріоритетом (спочатку високий пріоритет);
3) виконання тестів у порядку пріоритету;
4) при виявленні критичного дефекту можливе зупинення виконання менш пріоритетних тестів. Це дозволяє скоротити час отримання зворотного зв'язку про критичні проблеми.

Таким чином, математичне та алгоритмічне забезпечення системи тестування включає математичні моделі для оцінки ефективності тестування (покриття коду, ймовірність виявлення дефектів), алгоритми генерації тестових даних, моделі оцінки продуктивності (час відповіді, пропускна здатність), алгоритми обробки помилок, алгоритми автентифікації та авторизації, а також алгоритми генерації звітів та пріоритезації тестів. Ці моделі та алгоритми забезпечують наукову обґрунтованість та ефективність розробленої системи тестування.

Висновки до розділу II

У другому розділі було проведено детальне проектування інформаційного, архітектурного та математичного забезпечення системи тестування. На основі аналізу предметної області визначено структуру вхідних та вихідних даних для трьох ключових компонентів: серверної частини (FastAPI), веб-інтерфейсу (Vue.js) та інтеграційного сервісу (Telegram-бот). Встановлено, що ефективність тестування критично залежить від якості генерації тестових даних,

використання фікстур та забезпечення повної ізоляції тестів через механізми транзакційності бази даних.

Проектування архітектури системи базується на принципах модульності та багаторівневості. Розроблено структуру тестової інфраструктури, що включає використання патернів Page Object для веб-тестів та Factory/Fixture для управління станом даних. Описана ER-модель тестової бази даних та UML-структура класів забезпечують цілісність процесу тестування, а інтеграція в CI/CD конвеєр дозволяє автоматизувати перевірку якості на кожному етапі розробки — від окремих функцій до комплексних сценаріїв взаємодії користувача з системою.

Математичний апарат розділу охоплює метрики покриття коду (Line, Branch, Function\ Coverage) та ймовірнісні моделі виявлення дефектів, що дозволяє кількісно оцінити надійність програмного продукту. Окрему увагу приділено алгоритмам обробки результатів та моделям оцінки продуктивності, таким як теорія масового обслуговування (M/M/1) та аналіз процентилів часу відповіді (P95, P99).

Алгоритмічне забезпечення, що включає методи безпечного хешування (bcrypt), генерації JWT-токенів та автоматизованої валідації вхідних параметрів, гарантує захищеність системи та стійкість до типових вразливостей. Сформоване в цьому розділі теоретичне та алгоритмічне підґрунтя є вичерпною базою для переходу до практичної реалізації системи тестування в наступних частинах роботи.

РОЗДІЛ ІІІ. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Засоби розробки

Огляд інструментів для розробки тестової системи
Розробка комплексної системи тестування для багатокомпонентного веб-додатку управління розкладом вимагає використання сучасного набору інструментів, що забезпечують ефективну розробку, тестування та впровадження тестової інфраструктури. Вибір інструментів базується на технологічному стеці досліджуваної системи, вимогах до продуктивності та масштабованості, а також необхідності забезпечення надійності та стабільності тестових процесів.

3.1.1. Ядро розробки та тестування(Backend & Bot)

Основною мовою програмування для серверної частини та Telegram бота є Python, що обґрунтовано широкими можливостями для автоматизації тестування, великою екосистемою бібліотек та підтримкою асинхронного програмування. Python є ідеальним вибором для тестування API завдяки таким бібліотекам як pytest, requests, httpx, testcontainers, що забезпечують гнучкість та потужність для створення різних типів тестів.
Pytest є основним фреймворком для тестування в Python, що підтримує написання тестів різних рівнів, включаючи unit, integration та functional тести. Переваги pytest включають простий синтаксис написання тестів, потужну систему фікстур для управління тестовими даними, підтримку паралельного виконання тестів, детальні звіти про покриття коду та велику екосистему плагінів для розширення функціональності. Pytest підтримує параметризовані тести, що дозволяє виконувати один тест з різними наборами вхідних даних, що є критично важливим для комплексного тестування API.

Requests є стандартною бібліотекою для виконання HTTP-запитів в Python, що використовується для тестування API endpoints. Бібліотека підтримує всі основні HTTP методи (GET, POST, PUT, DELETE), дозволяє налаштовувати заголовки, параметри запитів, тіло запитів та обробляти відповіді. Requests має простий та інтуїтивно зрозумілий API, що спрощує написання тестів для API та робить код більш читабельним.

Testcontainers є інструментом для створення ізольованих тестових середовищ з використанням Docker контейнерів. Цей інструмент дозволяє автоматично запускати тестові бази даних (PostgreSQL), веб-сервери та інші сервіси в контейнерах, що забезпечує повну ізоляцію тестів та відтворення

реального середовища розгортання. Testcontainers підтримує різні бази даних, включаючи PostgreSQL, MySQL, MongoDB, що дозволяє тестувати інтеграцію з різними системами зберігання даних.

Python-telegram-bot є офіційною бібліотекою для розробки Telegram ботів, що використовується для створення тестової інфраструктури для тестування логіки бота. Бібліотека підтримує всі основні функції Telegram API, включаючи обробку повідомлень, callback-кнопок, inline-кнопок та вебхуків. Для тестування використовуються моки та стаби, що імітують поведінку Telegram API без реальної інтеграції з месенджером.

3.1.2. Тестування Frontend

Для тестування веб-інтерфейсу, розробленого на Vue.js, використовується сучасний інструмент Playwright, що забезпечує надійне та швидке тестування end-to-end сценаріїв. Playwright підтримує всі основні браузери (Chrome, Firefox, Safari, Edge) через єдиний API, що спрощує крос-браузерне тестування.
Playwright має переваги порівняно з традиційними інструментами, такими як Selenium: вищу швидкість виконання тестів завдяки використанню протоколу DevTools, кращу стабільність тестів через автоматичне очікування елементів, вбудовані можливості для тестування мобільних браузерів та зручні інструменти для налагодження тестів. Playwright підтримує TypeScript та JavaScript, що дозволяє писати тести з типізацією та зменшує кількість помилок під час розробки.

Vue Test Utils є офіційною бібліотекою для unit-тестування Vue компонентів, що забезпечує можливість монтування компонентів в ізольованому середовищі, симуляції подій, перевірки стану компонентів та взаємодії з ними. Ця бібліотека інтегрується з тестовими фреймворками, такими як Jest або Vitest, що дозволяє використовувати знайомий синтаксис та інструменти для тестування компонентів Vue.js.

Vitest є сучасним тестовим фреймворком для JavaScript та TypeScript, що оптимізований для Vue.js проєктів. Vitest забезпечує швидке виконання тестів завдяки використанню ES modules, вбудовану підтримку TypeScript, гарну інтеграцію з Vue Test Utils та зручний інтерфейс для запуску тестів в режимі watch.

3.1.3. Спеціалізоване тестування(Навантаження та Безпека)

Інструменти для performance та load тестування

Для тестування продуктивності та навантаження використовується K6 - сучасний інструмент для load testing, що використовує JavaScript для написання тестових сценаріїв. K6 підтримує різні типи навантаження, включаючи stress,

spike, soak та surge тестування, що дозволяє повноцінно оцінити поведінку системи в різних умовах.

K6 має переваги порівняно з традиційними інструментами, такими як JMeter: простий синтаксис написання тестів, вбудовані метрики продуктивності, гарну інтеграцію з хмарними сервісами для розподіленого тестування та можливість генерації звітів у різних форматах (JSON, HTML, InfluxDB). K6 підтримує різні протоколи, включаючи HTTP, WebSocket, gRPC, що дозволяє тестувати різні аспекти системи.

Інструменти для security тестування

Для тестування безпеки програмного забезпечення використовується OWASP ZAP - безкоштовий інструмент для автоматичного сканування веб-додатків на вразливості. ZAP підтримує різні типи сканувань, включаючи пасивне сканування, активне сканування, fuzz-тестування та сканування відповідно до OWASP Top 10.

OWASP ZAP дозволяє виявляти такі вразливості як SQL Injection, Cross-Site Scripting (XSS), Broken Authentication, Sensitive Data Exposure, XML External Entities (XXE), Broken Access Control, Security Misconfigurations, Cross-Site Request Forgery (CSRF), Using Components with Known Vulnerabilities, Insufficient Logging & Monitoring та Insecure Deserialization. Інструмент має зручний веб-інтерфейс для аналізу результатів сканування та API для інтеграції в автоматизовані процеси тестування.

3.1.4. Інфраструктура та CI/CD

Для автоматизації процесів розробки та тестування використовується GitHub Actions - платформа для безперервної інтеграції та безперервного розгортання, що інтегрується безпосередньо з репозиторіями GitHub. GitHub Actions дозволяє автоматизувати запуск тестів при кожному коміті, створенні pull request або за розкладом.

GitHub Actions підтримує різні операційні системи (Ubuntu, Windows, macOS), дозволяє використовувати Docker контейнери для ізоляції тестового середовища, має велику бібліотеку готових actions для поширених завдань та забезпечує гнучке налаштування робочих процесів. Для тестової системи використовується workflow, що автоматично запускає unit тести при кожному push, integration тести при створенні pull request та повний набір тестів перед релізом.

Docker є платформою для контейнеризації додатків, що використовується для створення ізольованих тестових середовищ. Docker дозволяє пакувати тестову інфраструктуру в контейнери, що забезпечує відтворення однакового

середовища на різних машинах розробників, тестових серверів та продакшн серверів.

Docker Compose використовується для управління кількома контейнерами, що складають тестову інфраструктуру: PostgreSQL база даних, FastAPI сервер, Vue.js додаток, Telegram бот. Це дозволяє одним командою запускати повне тестове середовище, що є критично важливим для integration та E2E тестування.

3.1.5. Звітність та моніторинг

Для генерації звітів про покриття коду використовується pytest-cov - плагін для pytest, що збирає статистику про виконання рядків коду та генерує детальні звіти в різних форматах. pytest-cov підтримує HTML звіти з інтерактивною навігацією по файлах, текстові звіти для консолі та XML звіти для інтеграції з CI/CD системами.

Allure є фреймворком для створення детальних та інтерактивних звітів про результати тестування. Allure підтримує різні типи тестів, включаючи unit, integration, E2E, та дозволяє групувати тести за функціональними блоками, тегами та пріоритетами. Звіти Allure включають графіки проходження тестів, історію виконання, скріншоти при невдачах та детальні логи виконання.

Таким чином, обраний набір інструментів для розробки тестової системи забезпечує комплексний підхід до тестування всіх компонентів досліджуваної системи, включаючи API, веб-інтерфейс та Telegram бота. Використання сучасних інструментів дозволяє створити надійну, масштабовану та ефективну систему тестування, що відповідає вимогам сучасних розробок програмного забезпечення .

3.2. Вимоги до технічного та програмного забезпечення

3.2.1. Локально реалізація моделі OSI

Архітектура відкритих систем стандарту OSI для локального середовища

Для реалізації комплексної системи тестування на локальному комп'ютері використовується спрощена архітектура на базі моделі OSI, що забезпечує необхідну функціональність при мінімальних апаратних ресурсах.

Фізичний рівень (Layer 1) реалізований через локальні інтерфейси комп'ютера: Ethernet порт для підключення до локальної мережі, Wi-Fi адаптер для бездротового доступу, USB порти для підключення зовнішніх пристроїв. Цей рівень забезпечує фізичну передачу даних між компонентами системи на одному комп'ютері.

Канальний рівень (Layer 2) реалізований через локальні мережеві інтерфейси з підтримкою MAC-адресації. Для локальної розробки

використовується loopback інтерфейс (127.0.0.1) для з'єднання між сервісами на одному комп'ютері, що забезпечує швидку передачу даних без залучення фізичної мережі.

Мережевий рівень (Layer 3) реалізований через протоколи IPv4 та IPv6 з підтримкою локальної маршрутизації. Для локальної розробки використовуються локальні IP-адреси (127.0.0.1, ::1) для з'єднання між сервісами, що забезпечує ізоляцію від зовнішньої мережі та швидку комунікацію.

Транспортний рівень (Layer 4) реалізований через протоколи TCP та UDP для локальної комунікації. TCP використовується для HTTP/HTTPS запитів між сервісами на одному комп'ютері, що гарантує надійну доставку даних. UDP може використовуватися для локальних WebSocket з'єднань.

Сеансовий рівень (Layer 5) реалізований через локальні механізми управління сесіями. Для локальної розробки використовуються JWT токени для автентифікації між сервісами та локальні сесії бази даних, що забезпечують збереження стану між запитами.

Представницький рівень (Layer 6) реалізований через локальні формати даних. У локальному середовищі використовуються формати JSON для API даних, HTML для веб-інтерфейсу, текстові повідомлення для симуляції Telegram бота.

Прикладний рівень (Layer 7) реалізований через локальні протоколи та API. HTTP/HTTPS для локального веб-сервера, WebSocket для real-time комунікації, локальні API для симуляції Telegram Bot API.

3.2.2. Програмний стек системи (Персональний Комп’ютор)

Операційна система робочої станції - Windows 11, що використовується для розробки та тестування системи. Вибір обґрунтовано підтримкою необхідних інструментів розробки (VS Code, Git, Docker Desktop), сумісністю з існуючою інфраструктурою та зручністю використання.

Віртуалізація - Docker Desktop для контейнеризації сервісів. Docker дозволяє створювати ізольовані середовища для кожного компонента системи (API, база даних, веб-інтерфейс, Telegram бот) на одному комп'ютері, що забезпечує відтворення реальних умов розгортання.

Мережна СУБД - PostgreSQL 14.x в Docker контейнері. PostgreSQL обґрунтовано підтримкою ACID транзакцій, розширеними можливостями JSON, потужною системою індексування. Використання Docker забезпечує легкість розгортання та ізоляцію від основної системи.

СУБД АРМів - SQLite для локального unit-тестування. SQLite забезпечує швидкість та ізоляцію для unit-тестів без необхідності налаштування повноцінної бази даних, що ідеально підходить для локальної розробки.

Інструментальне програмне забезпечення

Тестові фреймворки:

Pytest 7.x для автоматизованого тестування Python коду з підтримкою параметризації, фікстур та плагінів

Playwright 1.40+ для E2E тестування веб-інтерфейсу з підтримкою всіх основних браузерів

Vitest 1.x для unit тестування Vue компонентів з підтримкою TypeScript

K6 0.50+ для load testing з підтримкою JavaScript скриптів

Інструменти розробки:

WindSurf(VS code) з розширеннями для Python, Vue.js, Docker

Git для контролю версій з локальним репозиторієм

Docker Desktop для управління контейнерами

Node.js 18+ для Vue.js розробки

Python 3.11+ для API та бота

Інструменти CI/CD:

GitHub Actions для автоматизації процесів (хоча виконуються локально)

Docker Compose для оркестрації локальних контейнерів

npm scripts для автоматизації локальних процесів

Функціональне програмне забезпечення

Системне програмне забезпечення включає ядро Windows 11, драйвери обладнання, системні сервіси та утиліти управління. Системне ПЗ забезпечує базову функціональність для виконання тестових фреймворків та інструментів.

Інструментальне програмне забезпечення включає набір інструментів для локальної розробки та тестування. Ці інструменти забезпечують автоматизацію всіх етапів життєвого циклу тестування від написання тестів до генерації звітів.

Функціональне програмне забезпечення включає розроблену систему тестування з наступними компонентами:

Тестовий фреймворк для API з підтримкою різних типів тестів

Тестовий фреймворк для веб-інтерфейсу з підтримкою локального тестування

Тестовий фреймворк для Telegram бота з підтримкою моків та стабів

Система звітності про результати тестування

Інструменти для моніторингу та аналізу продуктивності

3.2.3. Локальна топологія та Docker-оркестрація

Тестова система розгорнута на одному комп'ютері з використанням Docker контейнерів для ізоляції компонентів. Топологія включає наступні компоненти:

Локальні сервіси:
API сервер на порту 8000 (localhost:8000)
Веб-інтерфейс на порту 3000 (localhost:3000)
PostgreSQL база даних на порту 5432 (localhost:5432)
Telegram бот симулятор на порту 8080 (localhost:8080)
Docker контейнери:
Контейнер з PostgreSQL 14.x
Контейнер з FastAPI додатком
Контейнер з Nginx для статичних файлів
Контейнер для тестового середовища
Локальна мережа Docker:
Міст (bridge) мережа для з'єднання контейнерів
IP діапазон: 172.17.0.0/16
Ізоляція від зовнішньої мережі
Апаратні вимоги для локальної розробки
Мінімальні вимоги:
CPU: 4 ядра (Intel i5 або AMD Ryzen 5)
RAM: 16 GB DDR4
Storage: 500 GB SSD
Операційна система: Windows 10/11 або macOS/Linux
Рекомендовані вимоги:
CPU: 8 ядер (Intel i7 або AMD Ryzen 7)
RAM: 32 GB DDR4
Storage: 1 TB NVMe SSD
Операційна система: Windows 11 з WSL2 або Linux
Специфічні вимоги для тестування:
Підтримка віртуалізації (Intel VT-x або AMD-V)
Достатньо ресурсів для одночасного запуску 4-6 Docker контейнерів
Швидкий SSD для зменшення часу запуску тестів
Стабільне інтернет-з'єднання для зовнішніх сервісів
Специфічні вимоги для локального тестування
Вимоги до продуктивності:
Час запуску Docker контейнерів: < 30 секунд
Час виконання unit-тестів: < 2 хвилини
Час виконання integration-тестів: < 5 хвилин
Час виконання E2E тестів: < 10 хвилин
Вимоги до ізоляції:
Кожен тест повинен виконуватися в ізольованому середовищі
Тестові дані не повинні впливати на інші тести

Можливість паралельного виконання тестів
Автоматичне очищення тестових даних
Вимоги до зручності розробки:
Швидке перезавантаження сервісів при змінах коду
Зручна система логування для налагодження
Інтеграція з IDE для швидкого запуску тестів
Можливість відлагодження тестів в реальному часі

Таким чином, вимоги до технічного та програмного забезпечення для локальної розробки на одному ПК визначають мінімальну необхідну конфігурацію, програмне забезпечення та інструменти для ефективної розробки та тестування системи управління розкладом.

3.3. Опис програмної реалізації

3.3.1. Структура програмної реалізації

Розроблена система тестування має модульну архітектуру, що забезпечує гнучкість, масштабованість та легкість підтримки. Основна структура програми включає наступні компоненти: тестовий фреймворк для API, тестовий фреймворк для веб-інтерфейсу, тестовий фреймворк для Telegram бота, систему управління тестовими даними, систему звітності та систему моніторингу.

Тестовий фреймворк для API

Основний клас APITestFramework є центральним компонентом системи тестування API. Цей клас забезпечує базову функціональність для всіх API тестів, включаючи управління тестовим клієнтом, налаштування тестового середовища, автентифікацію та обробку результатів тестування.

Методи класу APITestFramework:

- setup() - ініціалізація тестового середовища, створення тестового клієнта, налаштування бази даних
- teardown() - очищення тестового середовища, закриття з'єднань, видалення тестових даних
- authenticate(email, password) - виконання автентифікації та отримання JWT токена
- create_test_data() - створення тестових даних (користувачів, груп, викладачів, розкладів)
- cleanup_test_data() - видалення тестових даних після виконання тестів
- Клас TestClient забезпечує виконання HTTP-запитів до API:
- get(endpoint, params) - виконання GET запиту з параметрами
- post(endpoint, data) - виконання POST запиту з даними

- put(endpoint, data) - виконання PUT запиту з даними
- delete(endpoint) - виконання DELETE запиту
- assert_status_code(response, expected_code) - перевірка статус-коду відповіді
- assert_json_schema(response, schema) - перевірка структури JSON відповіді

Вхідні дані: HTTP метод, URL endpoint, параметри запиту, тіло запиту, заголовки **Вихідні дані:** HTTP відповідь, статус-код, тіло відповіді, заголовки Принципи функціонування: Клас використовує бібліотеку requests для виконання HTTP-запитів, автоматично додає JWT токен до заголовків, обробляє помилки та повертає стандартизовану відповідь.

Тестовий фреймворк для веб-інтерфейсу

Основний клас WebTestFramework забезпечує функціональність для E2E тестування веб-інтерфейсу. Цей клас використовує Playwright для автоматизації браузера та взаємодії з веб-елементами.

Методи класу WebTestFramework:

- setup() - запуску браузера, налаштування контексту, відкриття сторінки
- teardown() - закриття браузера, збереження скріншотів, очищення
- navigate_to(url) - перехід на вказану URL
- fill_form(selector, value) - заповнення полів форми
- click(selector) - клік по елементу
- wait_for_element(selector) - очікування появи елемента
- assert_element_visible(selector) - перевірка видимості елемента
- assert_text_contains(selector, text) - перевірка тексту в елементі
- Клас PageObject інкапсулює взаємодію з конкретними сторінками:
- LoginPage - методи для роботи зі сторінкою логіну
- SchedulePage - методи для роботи зі сторінкою розкладу
- ProfilePage - методи для роботи зі сторінкою профілю
- AdminPage - методи для роботи з адміністративною панеллю

Вхідні дані: селектори CSS/XPath, значення для форм, URL сторінок Вихідні дані: стан елементів, текстове содержиме, скріншоти, логи браузера Принципи функціонування: Клас використовує Playwright API для управління браузером, автоматично очікує завантаження елементів, обробляє таймаути та зберігає результати тестування.

Тестовий фреймворк для Telegram бота

Основний клас BotTestFramework забезпечує функціональність для тестування Telegram бота. Цей клас використовує моки для симуляції Telegram API без реальної інтеграції з месенджером.

Методи класу BotTestFramework:

- setup() - створення моків для Telegram API, ініціалізація тестового бота

- teardown() - очищення моків, закриття тестового середовища
- simulate_message(user_id, text) - симуляція текстового повідомлення від користувача
- simulate_callback(user_id, callback_data) - симуляція натискання callback-кнопки
- simulate_inline_query(user_id, query) - симуляція inline-запиту
- assert_reply_contains(text) - перевірка тексту відповіді бота
- assert_button_exists(text) - перевірка наявності кнопки

Клас TelegramMock імітує поведінку Telegram API:

- send_message(chat_id, text, reply_markup) - імітація відправки повідомлення
- answer_callback_query(callback_id, text) - імітація відповіді на callback
- get_chat_member(chat_id, user_id) - імітація отримання інформації про користувача

Вхідні дані: ID користувача, текст повідомлення, callback дані, inline-запити Вихідні дані: відповіді бота, надіслані повідомлення, виклики API

Принципи функціонування: Клас використовує моки для перехоплення викликів Telegram API, повертає predefined відповіді, що дозволяє тестувати логіку бота без залежності від зовнішніх сервісів.

Система управління тестовими даними

Клас TestDataManager забезпечує створення та управління тестовими даними для всіх компонентів системи.

Методи класу TestDataManager:

- create_user(role, **kwargs) - створення тестового користувача з вказаною роллю
- create_group(**kwargs) - створення тестової групи
- create_teacher(**kwargs) - створення тестового викладача
- create_schedule(**kwargs) - створення тестового розкладу
- create_assignment(**kwargs) - створення тестового завдання
- cleanup_all() - видалення всіх тестових даних
- get_random_data(model_class) - отримання випадкових тестових даних

3.3.2. Ключові технічні рішення

Клас DataFactory використовує шаблон Factory Method для створення тестових об'єктів:

- UserFactory - створення користувачів з валідними даними
- GroupFactory - створення груп з унікальними назвами
- TeacherFactory - створення викладачів з реалістичними даними
- ScheduleFactory - створення розкладів з пов'язаними об'єктами

Вхідні дані: тип об'єкта, параметри створення, кількість об'єктів Вихідні дані: створені об'єкти з ID, унікальні дані для тестів Принципи функціонування: Система використовує SQLAlchemy ORM для роботи з базою даних, забезпечує унікальність даних через генерацію випадкових значень, автоматично керує зв'язками між об'єктами.

Система звітності

Клас TestReporter забезпечує збір та генерацію звітів про результати тестування.

Методи класу TestReporter:

- collect_results(test_results) - збір результатів виконання тестів
- generate_html_report() - генерація HTML звіту з графіками
- generate_json_report() - генерація JSON звіту для CI/CD
- generate_coverage_report() - генерація звіту про покриття коду
- send_notification(results) - відправка сповіщень про результати

Клас MetricsCollector збирає метрики виконання тестів:

- collect_execution_time() - збір часу виконання тестів
- collect_memory_usage() - збір використання пам'яті
- collect_cpu_usage() - збір використання CPU
- collect_database_queries() - збір кількості запитів до бази даних

Вхідні дані: результати тестів, метрики виконання, логи Вихідні дані: звіти в різних форматах (HTML, JSON, XML), графіки, сповіщення Принципи функціонування: Система використовує pytest плагіни для збору результатів, Allure для генерації інтерактивних звітів, pytest-cov для вимірювання покриття коду.

Інтеграція між компонентами

Клас TestOrchestrator забезпечує координацію роботи всіх компонентів тестової системи.

Методи класу TestOrchestrator:

- run_all_tests() - виконання повного набору тестів
- run_api_tests() - виконання лише API тестів
- run_web_tests() - виконання лише веб-тестів
- run_bot_tests() - виконання лише тестів бота
- generate_comprehensive_report() - генерація звіту по всіх тестах

Методи інтеграції:

- setup_test_environment() - налаштування всіх сервісів
- cleanup_test_environment() - очищення всіх сервісів
- parallel_execution() - паралельне виконання тестів
- error_handling() - централізована обробка помилок

Вхідні дані: конфігурація тестів, набори тестів, параметри виконання Вихідні дані: узагальнені результати тестування, звіти, метрики Принципи функціонування: Оркестратор використовує Docker Compose для управління сервісами, забезпечує послідовність виконання тестів, обробляє залежності між тестами.

Посилання на моделі та методи з розділу 2

Програмна реалізація використовує математичні моделі та алгоритми, описані в розділі 2:

- Модель покриття коду реалізована через pytest-cov для розрахунку Line Coverage, Branch Coverage та Function Coverage
- Алгоритми генерації тестових даних реалізовані в класах DataFactory з використанням правил валідації
- Моделі оцінки продуктивності реалізовані через K6 для вимірювання Response Time, Average Response Time та Throughput
- Алгоритми автентифікації реалізовані через JWT токени з bcrypt хешуванням паролів
- Система масового обслуговування реалізована для аналізу черг запитів та оптимізації продуктивності

Таким чином, опис програмної реалізації демонструє структуру розробленої системи тестування, включаючи основні класи, методи, вхідні та вихідні дані, а також принципи функціонування кожного компонента з посиланнями на математичні моделі та алгоритми з розділу 2.

3.4. Керівництво користувача

Опис використаних методів тестування

Розроблена система тестування використовує комплексний підхід до перевірки якості програмного забезпечення, що включає різні методи тестування для кожного компонента системи управління розкладом.

3.4.1. Unit-тестування (Модульне тестування)

Unit-тестування використовується для перевірки окремих функцій та класів в ізоляції від зовнішніх залежностей. Метод включає написання тестів для кожної функції з перевіркою коректності обробки вхідних даних, повернення очікуваних результатів та обробки виключних ситуацій. Для unit-тестування використовується фреймворк pytest з підтримкою моків (mocks) та стабів (stubs) для ізоляції залежностей.

3.4.2. Integration-тестування (Інтеграційне тестування)

Integration-тестування використовується для перевірки взаємодії між компонентами системи, включаючи інтеграцію з базою даних, зовнішніми API та іншими сервісами. Метод включає створення тестової бази даних, виконання реальних HTTP-запитів до API та перевірку коректності обробки транзакцій. Для integration-тестування використовуються Docker контейнери для створення ізольованого тестового середовища.

3.4.3. End-to-End (E2E) тестування (Наскрізне тестування)

End-to-End (E2E) тестування використовується для перевірки повних користувацьких сценаріїв у веб-інтерфейсі. Метод включає автоматизацію браузера для виконання дій користувача (навігація, заповнення форм, кліки по кнопках) та перевірку коректності відображення даних. Для E2E тестування використовується фреймворк Playwright з підтримкою крос-браузерного тестування.

3.4.4. Performance тестування (Навантажувальне тестування)

Performance тестування використовується для оцінки продуктивності системи під навантаженням. Метод включає симуляцію різних рівнів навантаження, вимірювання часу відповіді, пропускної здатності та використання ресурсів. Для performance тестування використовується інструмент K6 з підтримкою різних сценаріїв навантаження.

3.4.5. Security тестування (Тестування безпеки)

Security тестування використовується для виявлення вразливостей безпеки в системі. Метод включає автоматичне сканування на вразливості, тестування автентифікації та авторизації, перевірку на SQL Injection та XSS атаки. Для security тестування використовується інструмент OWASP ZAP.

Покрокова інструкція тестування

3.4.6. Запуск API тестів

Крок 1. Підготовка середовища:

```
cd OARestAPI
python -m venv venv
source venv/bin/activate  # для Windows: venv\Scripts\activate
pip install -r requirements.txt
```

Таблиця 1. Код для підготовки середовище АПІ

Крок 2. Налаштування бази даних:

```
export
DATABASE_URL=postgresql://test_user:test_password@localhost:5432/test
_schedule
python -m alembic upgrade head
```

Таблиця 2. Код для підготовки налаштування БД

Крок 3. Запуск unit-тестів:

```
pytest tests/unit/ -v --cov=src --cov-report=html
```

Таблиця 3. Код для запуск unit-тестів

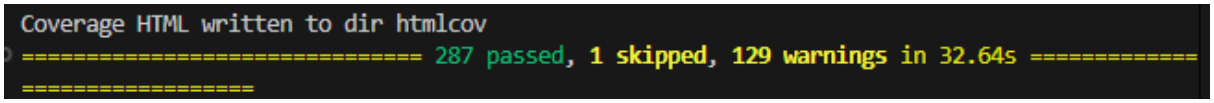


Рис. 3. Результат проходження юніт тестів

Крок 4. Запуск integration-тестів:

```
pytest tests/integration/ -v --cov=src --cov-report=html
```

Таблиця 4. Код для запуск integration-тестів

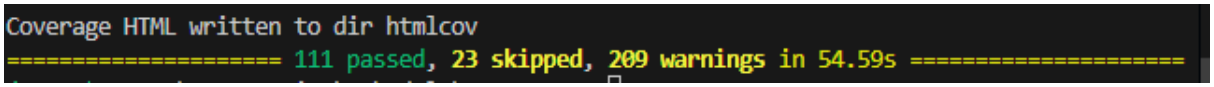


Рис. 4. Результат проходження інтеграційних тестів

Крок 5. Запуск E2E тестів:

```
pytest tests/e2e/ -v --cov=src --cov-report=html
```

Таблиця 5. Код для запуск E2E -тестів

Рис. 5. Результат проходження e2e тестів

Запуск всіх тестів в проєкті:

```
python -m pytest
```

Таблиця 6. Код для запуск всіх тестів в проєкті

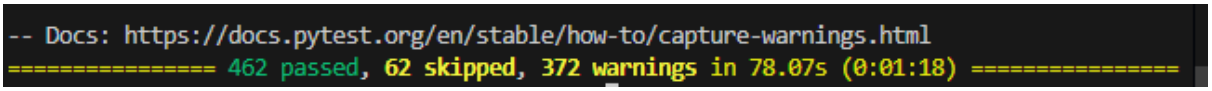


Рис. 6. Результат проходження юніт тестів

Перегляд звітів з покриття за шляхом ~/../htmlcov/index.html в браузері.

3.4.7. Тестування вебчастини

Крок 1. Налаштування змінних середовища:

```
cp .env.example .env.test
# Відредагувати .env.test з тестовими налаштуваннями
```

Таблиця 7. Код для налаштування змінного середовища для веб-тестів.

Крок 2. Запуск тестів:

```
npm run test:e2e
```

Таблиця 7. Код для запуск веб-тестів

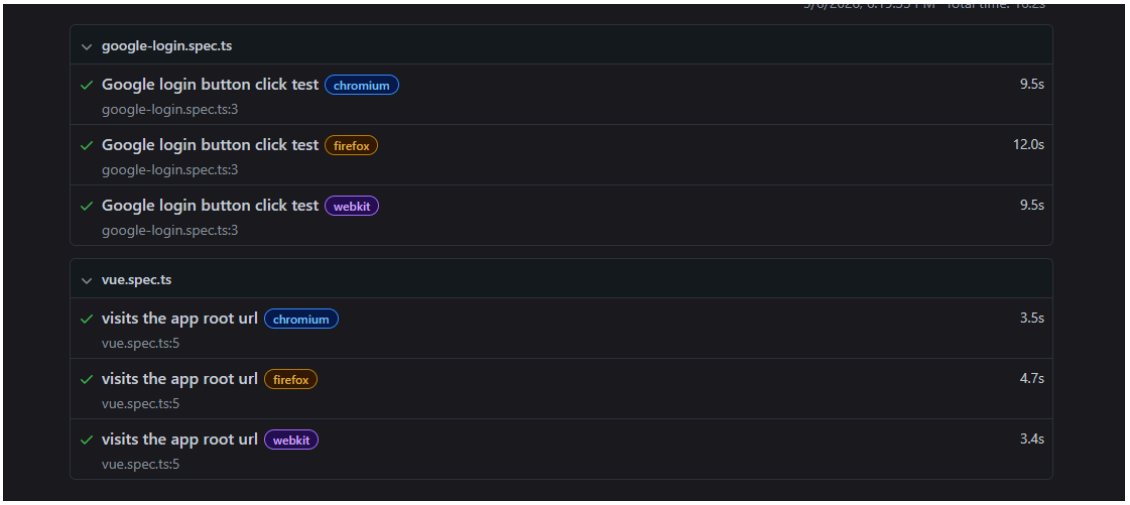


Рис.7. Результат проходження e2e веб-тестів

Першочергово тестування вебчастини планувалось проводитись в ручну. З описом , таблицями очікуваних результатів та критеріїв доступності. Автоматичні тести для вебсторінки першочергово перевіряє успішний сценарій входу де в змінних середовищах вводять валідні дані для входу користувача які будуть використовуватись pyright для введення даних та входу на сторінку(Виконання Login).

3.4.8. Тестування сторінки веб інструментами

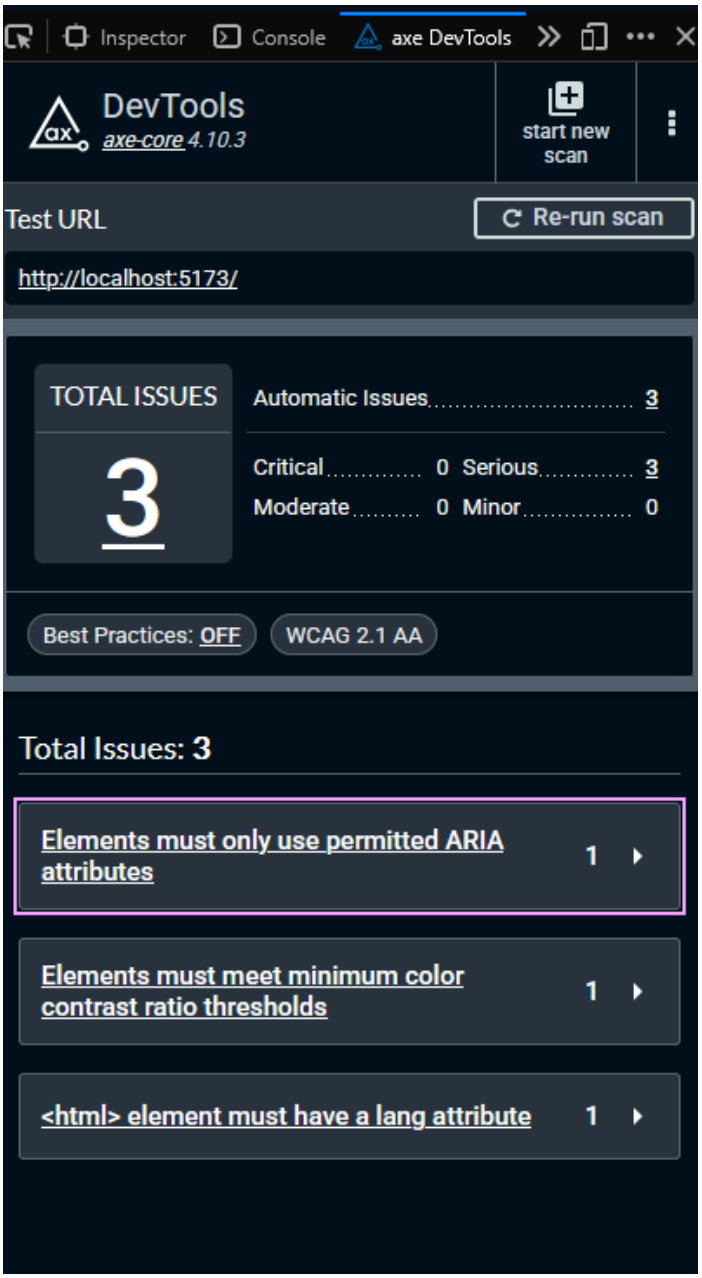


Рис. 7. Результат тестування сторінки через axeDevTools

Автоматизоване тестування доступності є першим етапом аудиту інтерфейсу, що дозволяє миттєво виявити найбільш критичні порушення стандартів WCAG 2.1 на рівні програмного коду. Для перевірки сторінки було використано професійне розширення axe DevTools, яке інтегрується безпосередньо в панель розробника браузера та проводить статичний аналіз DOM-дерева на відповідність правилам доступності.

На Рисунку 11 продемонстровано результати сканування головної сторінки системи управління розкладом. Згідно зі звітом, інструмент виявив 3 автоматичні проблеми (Automatic Issues), які мають статус Serious (серйозні). Це означає, що дані дефекти можуть суттєво ускладнити взаємодію з веб-сайтом для користувачів, що використовують допоміжні технології.

Аналіз виявлених дефектів:

Невідповідність ARIA-атрибутів (Elements must only use permitted ARIA attributes): Виявлено використання атрибутів, які не відповідають ролі елемента. Це може призвести до некоректної інтерпретації функціонала елемента скрінрідером.

Низька контрастність кольорів (Elements must meet minimum color contrast ratio thresholds): Один або декілька текстових елементів не відповідають мінімальному порогу контрастності для рівня AA. Згідно з методологією, для тексту цей показник має бути не менше 3:1 або 4.5:1 залежно від розміру шрифту.

Відсутність атрибута мови (element must have a lang attribute): Тег <html> не містить атрибута lang. Це критична помилка, оскільки екранні читачі (наприклад, NVDA) використовують цей атрибут для вибору правильної вимови та набору символів під час озвучування контенту.

Використання axe DevTools у поєднанні з подальшим мануальним тестуванням через NVDA та перевіркою навігації клавіатурою забезпечує комплексний підхід до оцінки доступності освітньої платформи. На основі отриманого звіту сформовано перелік рекомендацій для технічного виправлення виявлених зауважень.

3.4.9. Ручне тестування вебсторінки

Ручне тестування є критично важливим етапом перевірки якості програмного забезпечення, що доповнює автоматизоване тестування та дозволяє виявити проблеми, які можуть бути пропущені автоматизованими тестами. На відміну від автоматизованих тестів, що виконуються за predefined сценаріями, ручне тестування дозволяє дослідити поведінку системи з точки зору реального користувача, виявити UX проблеми, аномалії в інтерфейсі та аспекти доступності, що важко автоматизувати.

Тестування доступності (accessibility testing) є спеціалізованим напрямом ручного тестування, спрямованим на забезпечення доступності веб-інтерфейсів для людей з різними типами обмежень, включаючи візуальні порушення, порушення слуху, моторні порушення та когнітивні порушення. Доступність є критично важливою не лише з точки зору інклюзивності, але й згідно з нормативними вимогами та законодавством, що регулює використання інформаційних систем в освітніх закладах.

Для комплексної перевірки веб сторінки управління розкладом було обрано підхід, що включає послідовне виконання ручного тестування логіки функціонування кожної сторінки з подальшою перевіркою її доступності. Такий підхід дозволяє спочатку переконатися в коректності функціонування основних

сценаріїв використання, а потім перевірити, чи є ці сценарії доступними для всіх категорій користувачів.

Ручне тестування логіки включає перевірку коректності виконання основних користувацьких сценаріїв, валідації вхідних даних, відображення інформації та обробки помилок. Тестування доступності включає перевірку навігації клавіатурою, контрастності кольорів, підтримки скрінрідерів, адаптивності та відповідності стандартам WCAG 2.1 (Web Content Accessibility Guidelines).

Наступні розділи містять детальний опис результатів ручного тестування для кожної сторінки веб-інтерфейсу: сторінка входу, сторінка розкладу, сторінка профілю та адміністративна панель. Для кожної сторінки наведено як результати тестування логіки, так і результати перевірки доступності, що дозволяє сформувати комплексну оцінку якості та доступності веб-інтерфейсу.

3.4.9.1. Критерії оцінки:

Перевірка загальної логіки та взаємодії

Для визначення успіху чи невдачі буде використано слова маркери:

- Підтверджено - означає , що фактичний результат співпадає з очікуваним.
- Не підтверджено - означає, що фактичний результат не співпадає з очікуваним і буде задокументовано як саме.

Таблиця мануального тестування

В цій таблиці буде зафіксовано рух фокуса та доступність інтерактивних елементів сторінки для навігації лише клавіатурою.

Таблиця відповідності контрастності кольорів стандартам доступності

В цій таблиці буде зафіксовано дотримання вимог по контрастності на сторінці та проведено аналіз її доступності згідно настанов WCAG.

Для обчислення контрастності використовується формула

$(Contrast = L1 + 0.05 / L2 + 0.05)$

контрастність = (відносна яскравість світлого кольору) + 0.05 / (відносна яскравість темнішого кольору) + 0.05. Використовуватимуться зовнішні сервіси для порівняння та розрахунку коефіцієнта такі як: https://products.aspose.app/html/uk/contrast-checker

Контрастна вимога для AA - 3:1, AAA - 4.5:1

Результат тестування сторінки Screen Reader

В цьому пункті буде короткий опис того як інфомрація подоається з сторінки через інструменти з читання, а саме NVDA.

Ланцюг тестування:

Відкрити NVDA screen reader → виконати дії користувача на сторінці, по ітеруватись по елементах сторінки. → переглянути лог екранного читача → зробити висновки.

3.4.9.1.1. Сторінка входу

Сторінка входу є точкою доступу користувачів до системи управління розкладом. Сторінка призначена для аутентифікації користувачів через Google OAuth за наявності ключа @oa.edu.ua .

Структура сторінки:

- Заголовок - заголовок "Вітаємо в UniBot" з логотипом додатку
- Кнопка Google OAuth - кнопка "Продовжити через Google" для аутентифікації через Google. Та передає роль користувача в системі разом з його даними(JWT).
- Кнопка зміни теми сторінки з станом темна\світла
- Інтерактивна іконка “студент” яка змінюється при взаємодії.

Перевірка функціональності.

1. **Кнопка входу:**
    a. **Очікуваний результат**: користувач взаємодіє(клік) з кнопкою “Продовжити через Google” після чого підтверджує свою пошту через сервіси гугл і успішно входить в систему.
    b. **Фактичний результат:** Підтверджено.
2. **Кнопка зміни теми сторінки:**
    a. **Очікуваний результат:** користувач взаємодіє(клік) з кнопкою “Icon(moon)\Icon(sun)” після чого сторінка змінює палітру кольорів та зберігає в Local Storage обрану тему для цього пристрою.
    b. **Фактичний результат**: Підтверджено
    c.

Аналіз доступності сторінки:

Таблиця мануального тестування

В цій таблиці буде

ID елемента / Шлях	Очікуваний порядок	Фактичний порядок	Стан індикатора фокуса	Коментар
None → Кнопка входу	1	1	Чітка синя рамка фокусу навколо кнопки.	Відповідає вимогам доступності
None → кнопка зміни	2	3	Чітка синя рамка	Є похибка +1

теми			фокусу навколо кнопки.	ітеративної дії для переходу фокуса на кнопку від очікуваного результату

Таблиця 8. Результати мануального тестування сторінки входу

Таблиця відповідності контрастності кольорів стандартам доступності

Тип тексту	**Розмір (pt/px)**	**Стиль**	**Контраст AA**	**Контраст AAA**
Для світлої теми				
Текст вітання. Заголовок	28px	Передній план : Градієнт перший - #60a5fa, другий - #c084fc Колір фону: #fff	перший -2.43 другий - 2.52	перший -2.43 другий - 2.52
Загальний вміст сторінки. Інформаційне повідомлення.	15px	Передній план : #374151 Колір фону: #fff	10.31	10.31
Текстовий вміст кнопки “Продовжити через Google”	12px	Передній план : #0f172a Колір фону: #fff	17.85	17.85

Для темної теми				
Текст вітання. Загаловок	28px	Передній план : Градієнт перший - #60a5fa, другий - #c084fc Колір фону: #0f172a	перший -5.75 другий - 5.54	перший -5.75 другий - 5.54
Загальний вміст сторінки. Інформаційне повідомлення.	15px	Передній план : #f1f5f9 Колір фону: #1e293b	13.35	13.35
Текстовий вміст кнопки “Продовжити через Google”	12px	Передній план : #f1f5f9 Колір фону: #1e293b	13.35	13.35

Таблиця 9. Результати контрастності сторінки входу.

Підсумовуючи можемо сказати що сторінка входу є доступною за контрастність у більшості тестів аналізу, а саме оцінка 5\6. Проблема з читання може виникнути лише на світлій темі під час читання “Вітаємо в UniBot”, хоча цей текст є досить великим, що повинно компенсувати його контрастність

Результат тестування сторінки Screen Reader

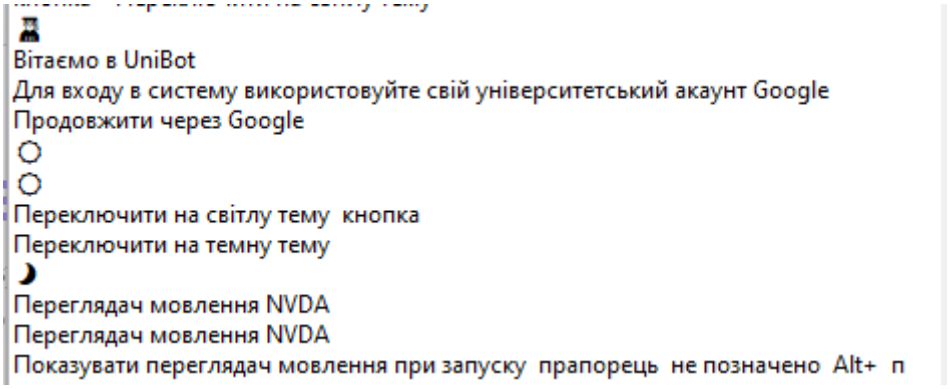


Рис. 8. Логи екранного читача про взаємодія читача з елементами сторінки.

Сторінка є повністю доступна за функціоналом для користування з допомоги екранного читача. Одне але, що інтерактивний елемент з Icon студент, не повідомляє користувачу про свою інтерактивність, хоча це ніяк не впливає на функціональну цінність. Також важливо зазначити, що хоча читач повідомляє про вміст кнопок на сторінці він не повідомляє, що це кнопка поки не буде взаємодії(клік) зі сторони користувача

3.4.9.1.2. HomePage. Сторінка Меню. Успішний вхід!

Сторінка Меню. Успішний вхід! (HomePage) є ключовим медіатором між сторінками. Саме на ній розміщені навігаційні кнопки та інформація про користувача в системі його пошта, роль, група, спеціальність. Сторінка призначена для навігації користувача по іншим розділам.

Структура сторінки

- Таблиця з інформацією про користувача пошта, роль, група, спеціальність

1. **Для ролі (Manager):**
   Кнопки навігації на сторінку\розділ:
   - Перейти до розкладу - переадресовує користувача на сторінку розкладу.
   - Управління заняттями - переадресовує користувача на сторінку управління заняттями як менеджера.
   - Управління парами - переадресовує користувача на сторінку управління парами як менеджера.
   - Вийти - переадресовує користувача на сторінку входу(LoginPage) при цьому завершує поточний сеанс користувача в системі.

Перевірка функціональності

1. **Кнопка переходу до розкладу:**
   a. **Очікуваний результат**: користувач взаємодіє(клік) з кнопкою після чого система переадресовує його на сторінку розкладу з його вхідними даними.
   b. **Фактичний результат:** Підтверджено.
2. **Кнопка управління заняттями:**
   a. **Очікуваний результат**: користувач взаємодіє(клік) з кнопкою після чого система переадресовує його на сторінку управління заняттями з його вхідними даними.
   b. **Фактичний результат:** Підтверджено.
3. **Кнопка управління парами:**

   a. **Очікуваний результат**: користувач взаємодіє(клік) з кнопкою після чого система переадресовує його на сторінку управління парами з його вхідними даними.
   b. **Фактичний результат:** Підтверджено.
4. **Кнопка управління парами:**
   a. **Очікуваний результат**: користувач взаємодіє(клік) з кнопкою після чого система переадресовує його на сторінку управління парами з його вхідними даними.
   b. **Фактичний результат:** Підтверджено.
5. **Кнопка вийти:**
   a. **Очікуваний результат**: користувач взаємодіє(клік) з кнопкою після чого система переадресовує його на сторінку входу в систему при цьому забуваючи його вхідні дані та закінчивши його сесію на сторінці.
   b. **Фактичний результат:** Підтверджено.

Аналіз доступності сторінки

Таблиця мануального тестування

Скорочення : Кнопка переходу до сторінки - КПдС

ID елемента / Шлях	Очікуваний порядок	Фактичний порядок	Стан індикатора фокуса	Коментар
None → КПдС розкладу	1	1	Чітка синя рамка фокусу навколо кнопки.	Відповідає вимогам доступності
КПдС розкладу → КПдС управління заняттями	1	1	Чітка синя рамка фокусу навколо кнопки.	Відповідає вимогам доступності
КПдС управління заняттями →	1	1	Чітка синя рамка фокусу навколо	Відповідає вимогам доступності

КПдС парами			кнопки.	
КПдС управління парами → Кнопка виходу з сесії.	1	1	Чітка синя рамка фокусу навколо кнопки.	Відповідає вимогам доступності
Кнопка виходу з сесії. → кнопка зміни теми	1	2	Чітка синя рамка фокусу навколо кнопки.	Є похибка +1 ітеративної дії для переходу фокуса на кнопку від очікуваного результату

Таблиця 10. Результати ручного тестування сторінки Home.

Таблиця відповідності контрастності кольорів стандартам доступності

Задля збереження лаконічності повторення перевірки схожих елементів які вже вносились до таблички не буде відбуватись, а лише посилюватиметься на попередні таблиці де це вже вказано. В таблиці нижче будуть вказані лише унікальні результати тестування які ще не зустрічались повторно в таблицях(Таблиця 18. Результати контрастності сторінки входу.).

Тип тексту	**Розмір (pt/px)**	**Стиль**	**Контраст AA**	**Контраст AAA**
Для світлої теми				
Текст про успіх.	28px	Передній план : Градієнт	перший -1.92 другий - 3.68	перший -1.92

Заголовок		перший - #34d399, другий - #3b82f6 Колір фону: #fff		другий - 3.68
Для темної теми				
Текст про успіх. Заголовок	28px	Передній план : Градієнт перший - #34d399, другий - #3b82f6 Колір фону: #1e293b	перший -7.61 другий - 3.98	перший -7.61 другий - 3.98

Таблиця 11. Результати контрастності сторінки Home

Підсумовуючи можемо сказати що домашня є доступною за контрастність у більшості тестів аналізу схожі результати на попередні з сторінки входу. Проблема з читання може виникнути лише на світлій темі під час читання повідомлення про успішний, хоча цей текст є досить великим, що повинно компенсувати його контрастність. Схоже було вже на сторінці входу.

Результат тестування сторінки Screen Reader

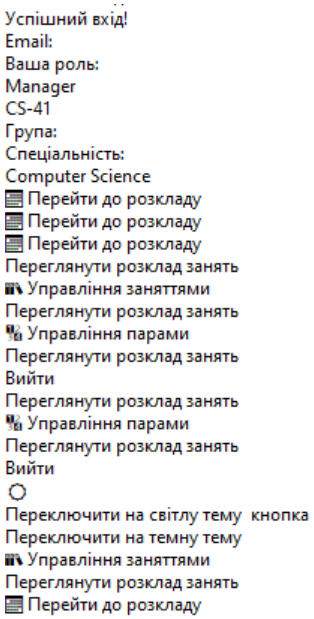


Рис. 9. Логи екранного читача про взаємодія читача з елементами сторінки.

Сторінка є повністю доступна за функціоналом для користування з допомоги екранного читача. Одне але, що інтерактивний елемент з Icon студент,

не повідомляє користувачу про свою інтерактивність, хоча це ніяк не впливає на функціональну цінність. Також важливо зазначити, що хоча читач повідомляє про вміст кнопок на сторінці він не повідомляє, що це кнопка поки не буде взаємодії(клік) зі сторони користувача

3.4.9.1.3. Сторінка розкладу.

Сторінка входу (Schedule Page) є інформаційної сторінкою для користувачів у системи управління розкладом. Сторінка призначена для інформування про заняття студентів, інформування про заняття, можливість внесення даних про заняття та швидке оцінювання викладачем та швидке керування змінами в розкладі менеджером.

Структура сторінки

- Хедер сторінки - текст про сторінку “Розклад занять” та кнопка “ ← На головну”.
- Календар днів - призначений для керування відображенням розкладу на конкретний вибраний період (день\тиждень)
- Конкретний розклад з парами на окремий день.
    - Інформація про заняття: номер пари, викладач, назва пари, тип пари та її номер, аудиторія, кнопка підключення до заняття в гугл міт.
    - **ОКРЕМО для юзерів з роллю “Вчитель” та “Менеджер розкладу”** кнопка “управління заняттям”.
        - Сама кнопка відкриває модальне вікно у якому:
            - Менеджер може змінити: дату проведення заняття, номер пари в розкладі та аудиторію.
            - Викладач може:
                - змінити\вказати: тему конкретного заняття та у вікні модального вікна “Тема заняття
                - виставити оцінку та відвідування студентам у розділі модального вікна “Журнал (оцінка/явка)
            - Сама форма має кнопку збереження внесених змін до системи.
- Кнопка вибору відображення розкладу за днем\тижнем.
- **ОКРЕМО для юзерів з роллю “User(учень)”**
    - Окремий розділ завдання в заняттям. З кнопкою “здати”.
        - Кнопка “здати” відкриває вікно “Здати завдання” де інформація про назву завдання, предмет, дедлайн, та саме завдання формату .md. В формі учень може :
            - Здати завдання для заняття.

Перевірка функціональності

1. **Кнопка “на головну”:**
    a. **Очікуваний результат:** користувач взаємодіє(клік) з кнопкою “ ← На головну після чого система його переадресовує на сторінку HomePage
    b. **Фактичний результат:** Підтверджено
2. **Календар:**
    a. **Очікуваний результат:** користувач взаємодіє(клік) з конкретним днем календар. Після чого перевіряється який тип відображення на сторінці обрано день\тиждень. Якщо обрано день відображається розклад на конкретний обраний день.
    якщо тиждень відображається розклад на конкретний тиждень в який входить обраний день понеділок-неділя.
    b. **Фактичний результат:** Підтверджено
3. **Кнопки навігації керування заняттями в розкладі:**
    a. **Кнопка інформації про викладача:**
        i. **Очікуваний результат**:користувач взаємодіє(клік) з іменем викладача на сторінці після чого відкривається модальне вікно з інформацією про: імя(повне ім'я), пошту та персональний код зустрічі(у заняття і викладача можуть бути різні коди зустрічі).
        ii. **Фактичний результат:** Підтверджено
    b. **Кнопка підключення до Google meet:**
        i. **Очікуваний результат:** користувач взаємодіє(клік) з кнопкою та система перенаправляє його на сторінку Google Meet за кодом зустрічі пари(у заняття і викладача можуть бути різні коди зустрічі).
        ii. **Фактичний результат:** Підтверджено
    c. **Кнопка управління заняттям**
        i. **Менеджер**
            1. **Форма редагування заняття:**
                a. **Очікуваний результат:** користувач взаємодіє з полями дати (клік) та змінює дату, взаємодіє(клік) з випадаючим меню “Номер пари” та змінює його, вводить у відповідне поле номер(Будь-що але валідне з наявних приміщень) після чого натискає на кнопку “Зберегти зміни розклад” і дані зберігаються до бази даних.
                b. **Фактичний результат:** Підтверджено
        ii. **Вчитель**

1. **Форма створення\зміни теми заняття:**
    a. **Очікуваний результат:** користувач вводить тему заняття в відповідне текстове поле. Після чого взаємодіє(клік) з кнопкою “Зберегти тему” і тема зберігається до бази даних.
    b. **Фактичний результат:** Підтверджено
2. **Форма ведення журналу оцінювання\явки:**
    a. **Очікуваний результат:** користувач взаємодіє(клік) з елементом switch та виставляє оцінку(числове значення) в відповідне поле після чого взаємодіє(клік) з кнопкою “Зберегти в журнал” і дані вносять в табличку бази даних.
    b. **Фактичний результат:** Підтверджено

d. **Розділ завдання для заняття. ДОСТУПНЕ ЛИШЕ ДЛЯ СТУДЕНТІВ**
    i. **Очікуваний результат:** Користувач вводить текст як відповідь у відповідне поле або\і завантажує файл з носія в систему якщо це потрібно у відповідному полі після взаємодії(клік) з кнопкою (“Browse..”) .
    ii. Після того як хоча б одне поле буде мати хоч щось, кнопка “Здати завдання” стане доступна для взаємодії(клік) після чого завдання буде збережно до бази даних. Також студент може просто вийти з форми взаємодією(клік) з кнопкою “Скасувати” або “X”.
    iii. **Фактичний результат:** Підтверджено

e. **Випадаюче меню фільтр. ДОСТУПНЕ ЛИШЕ ДЛЯ МЕНЕДЖЕРА**
    i. **Очікуваний результат:** користувач взаємодіє(клік) з випадаючим меню груп і вибирає конкретну з списку. Після чого розклад рендериться і відображає розклад для конкретної групи студентів.
    ii. **Фактичний результат:** Підтверджено

Аналіз доступності сторінки:

Таблиця мануального тестування

ID елемента / Шлях	**Очікуваний порядок**	**Фактичний порядок**	**Стан індикатора фокуса**	**Коментар**

None → Кнопка “на головну”	1	1	Чітка синя рамка фокусу навколо кнопки.	Відповідає вимогам доступності
Кнопка “на головну” → Перемикання місяця в календарику назад	1	1 і 3	Чітка синя рамка фокусу навколо кнопки.	Відповідає вимогам доступності
Перемикання місяця в календарику вперед → конкретний день календаря	1	1	Чітка синя рамка фокусу навколо кнопки.	Відповідає вимогам доступності
конкретний день календаря → кнопка перемикання відображення день\тиждень	1	1	Чітка синя рамка фокусу навколо кнопки.	Відповідає вимогам доступності
кнопка перемикання відображення день\тиждень → елемент заняття в розкладі	1	1	Чітка синя рамка фокусу навколо кнопки.	Відповідає вимогам доступності
елемент заняття в розкладі →	1	немає	не попадає в фокус	Не відповідає вимогам

кнопка інформації про вчителя				доступності
елемент заняття в розкладі → Посилання на гугл міт	2	1	Чітка синя рамка фокусу навколо кнопки.	Відповідає вимогам доступності
Елементи сторінки доступні лише Вчителю та Менеджеру				
Посилання на гугл міт → кнопка управління заняттям	1	1	Чітка синя рамка фокусу навколо кнопки.	Відповідає вимогам доступності
Вікно управління заняттями	Focus trap. Рух фокусу лише по елементам форми	Фокус виходить за межі форми	Чітка синя рамка фокусу навколо кнопки.	Не відповідає вимогам доступності
Елементи сторінки доступні лише Вчителю та Менеджеру				
Посилання на гугл міт → кнопка здачі заняття	1	1	Чітка синя рамка фокусу навколо кнопки.	Відповідає вимогам доступності
Вікно здачі заняття	Focus trap. Рух фокусу лише по елементам форми	Фокус виходить за межі форми	Чітка синя рамка фокусу навколо кнопки.	Не відповідає вимогам доступності

Таблиця 12. Результати ручного тестування сторінки Schedule.

Таблиця відповідності контрастності кольорів стандартам доступності

Тип тексту	**Розмір (pt/px)**	**Стиль**	**Контраст AA**	**Контраст AAA**
Для світлої теми				
Текст Розклад занять. Заголовок	24px	Передній план : #d1d5db Колір фону: #fff	5.67	5.67
Вибраний день в розкладі	14px	Передній план : #ffffff Колір фону: #059669	3.77	3.77
Інформаці про заняття номер, час закінчення, аудиторія	13px	Передній план : ##e2e8f0 Колір фону: #ffffff	1.23	1.23
Для темної теми				
Текст Розклад занять. Заголовок	24px	Передній план : градієнт: #60a5fa #c084fc Колір фону: #0f172a	7.02 6.76	7.02 6.76

Вибраний день в розкладі	14px	Передній план : #ffffff Колір фону: #059669	3.77	3.77
Інформаці про заняття номер, час закінчення, аудиторія	13px	Передній план : ##e2e8f0 Колір фону: #1e293b	11.87	11.87

Таблиця 13. Результати контрастності сторінки Рокзлад.

Підсумовуючи можемо сказати що сторінка розкладу має критичні помилки з контрастності у деяких тестах аналізу. Проблема читання в більшості випадків може виникнути під час користування світлою темою. Рекомендовано виправити контрастність кольорів на сторінці для кращого досвіду користувача.

Результат тестування сторінки Screen Reader

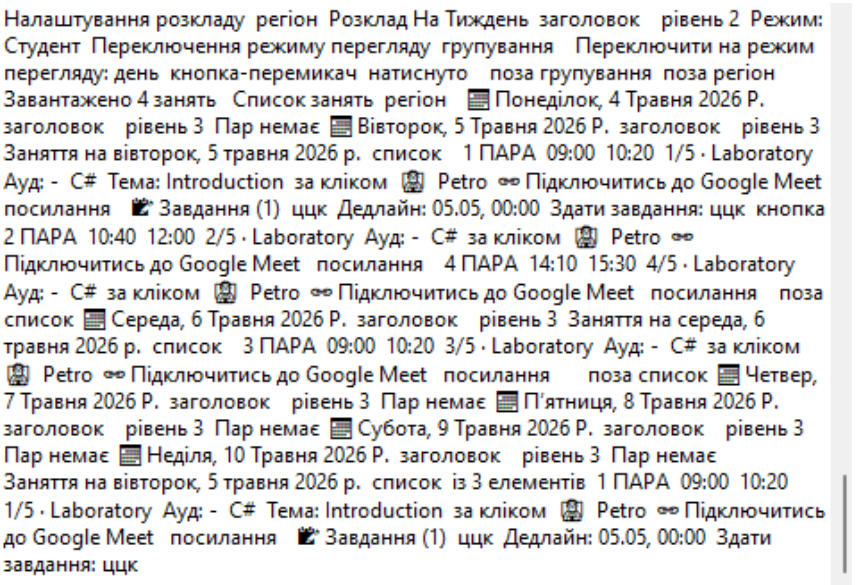


Рис. 10. Логи екранного читача про наявність елементів сторінки розкладу. (Студент)

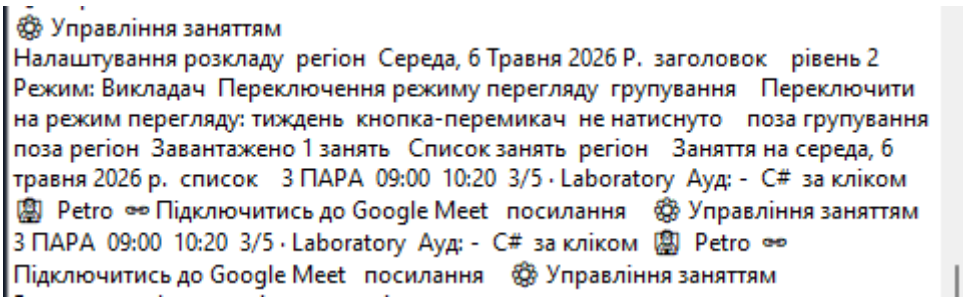


Рис. 11. Логи екранного читача про наявність елементів сторінки розкладу. (Вчитель)

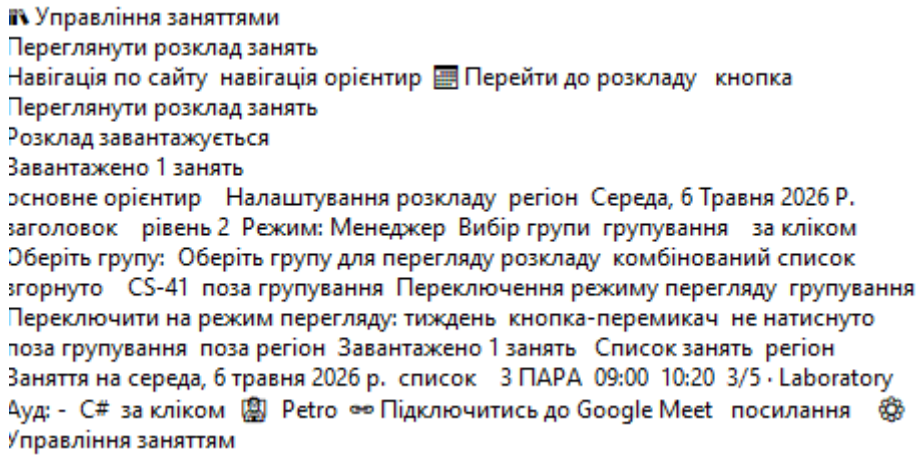


Рис. 12. Логи екранного читача про наявність елементів сторінки розкладу. (Менеджер)

3.4.9.1.4. Сторінка управління заняттями

Сторінка управління заняттям є точкою доступу користувачів з роллю “менеджер”до системи управління елементами розкладу:
– редагування інформації про заняття
–редагування загальних даних щодо організаційної навчальної бази: навальні предмети, навчальні групи, викладачі, типи занять.
Сторінка призначена для керування інформацією , що подається в системі розкладу.

Структура сторінки

- Хедер сторінки з заголовком“Управління заняттями”, функціональною кнопкою “+ Створити завдання” , та навігаційною кнопкою “ ← Назад”
- Табличка з переліком даних та статусом їхнього наповнення. З елементами якої можна взаємодіяти після чого на сторінці з’являється табличка з наповненням детальної інформації щодо даних. Елементи цієї таблички є інтерактивними , після взаємодії(клік) відкривається вікно на якому користувач може внести зміни в конкретний об’єкт таблички з його полями.
- Таблиця список занять, яка містить перелік всіх існуючих занять , що відображаються лише за активацією фільтрів. Після застосування фільтрів користувач отримує наповнення таблички заняттями розділених на типи (Лекції, Лабораторні, тощо). Також рядки таблички мають стовпчик “Дії” для редагування та видалення занять

Перевірка функціональності

1. **Кнопка “на головну”:**

a. **Очікуваний результат:** користувач взаємодіє(клік) з кнопкою “ ← На головну після чого система його переадресовує на сторінку HomePage
b. **Фактичний результат:** Підтверджено

2. **Кнопка “Створити заняття”:**
   a. **Очікуваний результат:** користувач взаємодіє(клік) з кнопкою “ + Створити заняття” після чого система відкриває модальне вікно з полями для створення заняття.
      i. **Вікно створення заняття:**
         1. **Очікуваний результат:** Користувач заповнює всі відповідні поля в модальному вікні. За потреби взаємодіє з елементами (клік) “+” де створює нові дані для створення заняття. Після заповнення всіх полів відповідними діями:
            a. –Дата: введення коректної дати в майбутньому;

            b. Пара обрана з списку
            c. Предмет обраний з списку
            d. Група обрана з списку
            e. Викладач обраний з списку
            f. Тип заняття обраний з списку
            g. Аудиторія вказується будь яке значення, що відповідає наявному приміщенню або формату навчання.
            h. Код підключення вказується автоматично за наявності коду у Викладача, якщо ж він відсутній або потрібен не особистий ключ вчителя то менеджером вводиться новий ключ в поле “Код підключення”
         2. **Фактичний результат:** Підтверджено
   b. **Фактичний результат:** Підтверджено

3. **Елемент статус даних:**
   a. **Очікуваний результат:** перегляд наявних об’єктів даних в таблиці та їхня кількість в бд. Можливість взаємодія(клік) на назву об’єкта для отримання детальної інформації про нього:
      1. **Очікуваний результат:** Створення детальної таблички про інформацію в об’єкті за взаємодією(клік) на нього:
         a. **Очікуваний результат:** відкриття модального вікна з відповідними полями обраного об’єкта з

можливістю редагування інформації та збереженням даних натиском(клік) на кнопку “Зберегти)

b. **Фактичний результат:** Підтверджено

2. **Фактичний результат:** Підтверджено

ii. **Фактичний результат:** Підтверджено

4. **Елемент список занять в таблиці**

a. Очікуваний результат: користувач обирає фільтр за яким отримує відповідне наповнення до таблички

i. Відображення розкладу занять з фільтрацією за одним або кількома фільтрами з можливістю взаємодії в з елементом в стовпчику таблички “Дії”

1. Обрано “Редагування заняття”. Відкриття модального вікна редагування, що заповнюється актуальними даними про заняття з можливістю внесення змін та збереженням взаємодією з елементами на вікні.
2. **Фактичний результат:** Підтверджено

ii. **Фактичний результат:** Підтверджено

b. **Фактичний результат:** Підтверджено

Аналіз доступності сторінки:

Таблиця мануального тестування

ID елемента / Шлях	**Очікуваний порядок**	**Фактичний порядок**	**Стан індикатора фокуса**	**Коментар**
None → Кнопка “Створення заняття”	1	1	Чітка синя рамка фокусу навколо кнопки.	Відповідає вимогам доступності
None → перший об’єкт даних з “Статус даних”	3	0	Не попадає в фокус. Як і всі інші елементи “Статусу даних”	Не відповідає вимогам доступност
None →	8	3	Чітка синя	Відповідає

перший фільтр розділу “Список занять”			рамка фокусу навколо кнопки.	вимогам доступності
Елементи фільтрів в розділі “Список занять	Фокус трап. Можливість ітеруватись по елементам списку	Підтверджено	Фокус коректно ітерується по кожному елементу списку	Відповідає вимогам доступності
Сторінка редагування заняття	Фокус трап. Фокус не виходить за межі модального вікна	Підтверджено	Чітка синя рамка фокусу навколо кожного елемента.	Відповідає вимогам доступності

Таблиця 14. Результати ручного тестування сторінки Управліннями.

Таблиця відповідності контрастності кольорів стандартам доступності

Тип тексту	**Розмір (pt/px)**	**Стиль**	**Контраст AA**	**Контраст AAA**
Для світлої теми				
Текст таблички розкладу занять в стовпчику “Група”	24px	Передній план : #8b5cf6 Колір фону: #e2e8f0	3.43	3.43

Для темної теми				
Текст таблички розкладу занять в стовпчику “Група”	24px	Передній план : #8b5cf6 Колір фону: #0f172a	4.22	4.22

Таблиця 15. Результати контрастності сторінки Управління заняттями

Підсумовуючи можемо сказати що сторінка входу є доступною за контрастність у більшості тестів аналізу. Проблема з читання може виникнути лише на світлій темі під час перегляду стовпчика “Група” з таблички розкладу занять.

Результат тестування сторінки Screen Reader

C# programming
Опис
Фільтри
Список занять
4 занять
4 занять
основне орієнтир
Група комбінований список Всі групи згорнуто
комбінований список розгорнуто
Всі групи список
Всі групи 1 із 3
Всі групи
Всі групи
CS-41 не виділено 2 із 3
CS-41
Група комбінований список CS-41 згорнуто
Предмет
Прогрес
Vite App документ
—
Petro
—
Petro
4 пара
05.05.2026
CS-41

Рис. 13. Логи екранного читача про наявність елементів сторінки розкладу. (Менеджер)

Сторінка є незрозуміла для екранного читача. Рекомендовано покращити описи елементів сторінки, особливо варто звернути увагу на інтерактивні кнопки, посилання, меню.

3.4.9.1.5. Сторінка журнал занять

Сторінка журнал занять(доступна лише Викладачеві) є кабінетом керування вчителя своїми дисциплінами. З можливостями: створення звіту щодо навантаженню по всіх дисциплінах; Ведення журналу в повноекранному режимі, перегляд, виставлення оцінок та відвідуваності студентам\учням;експортувати журнал як документ для збереження інформації на носій;

Структура сторінки

- Хедер в якому розміщено текст “Журнал занять” та кнопку “ ← На головну” яка переадресовує користувача на головну сторінку при взаємодії(клік).
- Секцію “Мої дисципліни” де розміщено перелік дисциплін викладача який зайшов в систему. Також в цій секції розміщена кнопка “Створення звітності” яка відкриває модальне вікно з можливість створити файл по звітності академічного навантаження на викладача.
- Секція “Дисципліна. Журнал успішності та відвідуваності для C# ”. В залежно від того яка дисципліна обрана в секції “Мої дисципліни” відповідно таке наповнення отримує табличка журналу.
- В самому журналі ми користувач може вносити зміни в відвідування конкретного учня\студента та оцінювання.
- Окрема інформаційна секція з позначками та пояснення емелентів сторінки. Розміщено кнопку “Шкала оцінок” після взаємодії(клік) відкривається модальне вікно на якому чітко розписано систему конвертації балів в оцінювання літерами та градацію самих оцінок.
- Секція назви дисципліни окрім назви де крім назви розміщено кнопку “Експорт даних на Гугл диск”, що експортує дані журналу на хмарний носій.

Перевірка функціональності

1. **Кнопка “на головну”:**
    a. **Очікуваний результат:** користувач взаємодіє(клік) з кнопкою “ ← На головну після чого система його переадресовує на сторінку HomePage
    b. **Фактичний результат:** Підтверджено
2. **Кнопка “Експорт таблиці академічного навантаження”**

   a. **Очікуваний результат:** користувач взаємодіє(клік) з кнопкою після чого на сторінці відкривається модальне вікно, з можливістю ввести назву і кнопками “Скасувати” та “Створити звіт”.
      i. Після взаємодії з кнопкою “Скасувати”. Вікно закривається.
      ii. Після взаємодії з кнопкою “Створити звіт” на гугл диску користувача автоматично створюється звіт з відповідною назвою вказаною на формі раніше. І вікно закривається.
   b. **Фактичний результат:** Підтверджено
3. **Кнопка “Експорт на Google Диск”**
   a. **Очікуваний результат:** на гугл диску створюється документ з журналу дисципліни під час перегляду якої відбулась взаємодія(клік) з кнопкою.
   b. **Фактичний результат:** Підтверджено
4. **Поле виставлення оцінки студенту**
   a. **Очікуваний результат:** користувач виставляє значення(число) у відповідне поле після чого натискає на “Enter”(Клавіша) або виходить фокусом на інший елемент, оцінка автоматично зберігається і в стовпчику “Загалом” підраховується загальна сума всіх балів враховуючи останній виставлений.
   b. **Фактичний результат:** Підтверджено
5. **Поле виставлення відвідування студенту**
   a. **Очікуваний результат:** користувач взаємодіє(клік) з червоною\зеленою кнопкою явки студента на парі в журналі. Після чого статус відвідування автоматично зберігається і в стовпчику “% відвідувань” підраховується загальний % відвідування враховуючи останні зміни.
   b. **Фактичний результат:** Підтверджено

Аналіз доступності сторінки:

Таблиця мануального тестування

ID елемента / Шлях	Очікуваний порядок	Фактичний порядок	Стан індикатора фокуса	Коментар
None → Кнопка “на головну”	1	1	Чітка синя рамка фокусу навколо кнопки.	Відповідає вимогам доступності

None → кнопка створення звіту академічного навантаження	2	2	Чітка синя рамка фокусу навколо кнопки.	Відповідає вимогам доступності
Пастка фокусу в формі “Створення звітності”	Фокус рухається лише по елементам форми	Підтверджено	Чітка синя рамка фокусу навколо кнопки.	Відповідає вимогам доступності
кнопка створення звіту академічного навантаження→ перша дисципліна з списку мої дисципліни	1	1	Чітка синя рамка фокусу навколо кнопки.	Відповідає вимогам доступності
перша дисципліна з списку мої дисципліни → кнопка “експорт на Google Диск”	1	1	Чітка синя рамка фокусу навколо кнопки.	Відповідає вимогам доступності
кнопка “експорт на Google Диск”→ Перший елемент журналу	1	1	Чітка синя рамка фокусу навколо кнопки.	Відповідає вимогам доступності
Останній елемент журналу → Кнопка “Шкала оцінок”	1	1	Чітка синя рамка фокусу навколо кнопки.	Відповідає вимогам доступності

Таблиця 16. Результати ручного тестування сторінки Журнал занять вчителя.

Таблиця відповідності контрастності кольорів стандартам доступності

Тип тексту	Розмір (pt/px)	Стиль	Контраст AA	Контраст AAA
Для світлої теми				
форма для виставлення оцінки студенту	26px	Передній план : #9ca3af Колір фону: #ffffff	2.54	2.54
Для темної теми				
форма для виставлення оцінки студенту	26px	Передній план : #f1f5f9 Колір фону: #1e293b	13.35	13.35

Таблиця 17. Результати контрастності сторінки Журнал занять

Критичний елементи на цій сторінці є форма для виставлення оцінки студенту на світлій темі через її коефіцієнт контрастності. Важливо виправити це для кращого користувацького досвіду. На зараз користувачу який вперше на сторінці буде проблемо зрозуміти де саме ця форма, через розмитість її країв та загального фону сторінки.

Результат тестування сторінки Screen Reader

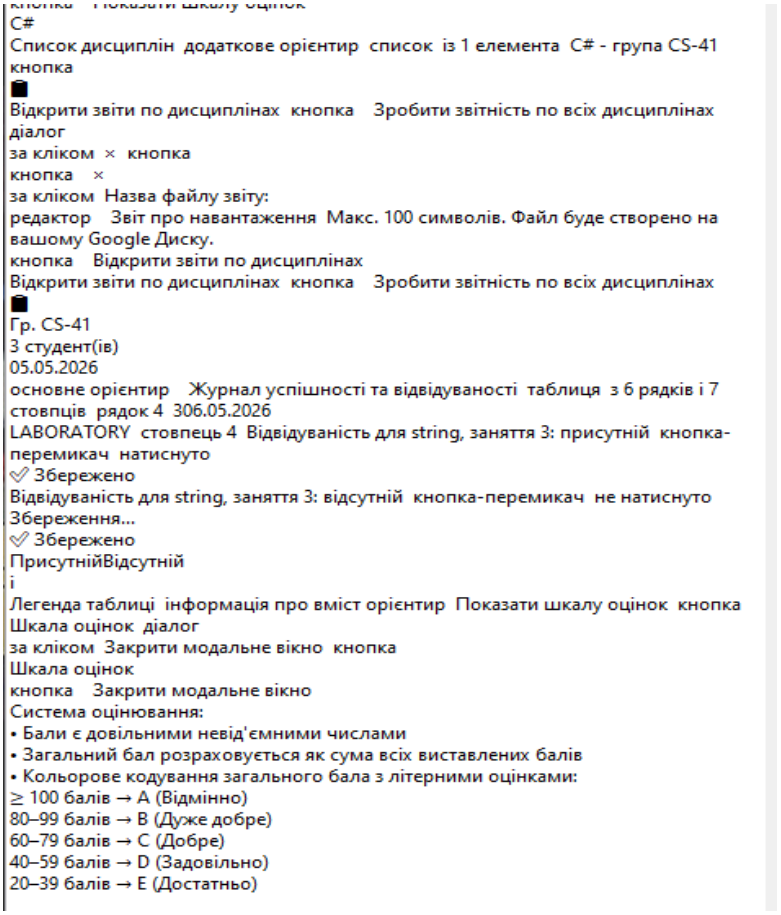


Рис. 14. Логи екранного читача про наявність елементів сторінки журналу занять.

Висновки до розділу III

У третьому розділі було успішно реалізовано програмне та технічне забезпечення комплексної системи тестування, побудованої на базі мови програмування Python 3.11+ та сучасних методів контейнеризації. Вибір інструментарію, що включає pytest для серверної логіки, Playwright для веб-інтерфейсу та K6 для аналізу навантаження, дозволив створити гнучку інфраструктуру, яка повністю відповідає технологічному стеку досліджуваної системи управління розкладом. Завдяки використанню Docker та Testcontainers було досягнуто повної ізоляції тестових середовищ, що гарантує відтворюваність результатів незалежно від апаратних особливостей робочої станції користувача.

Розроблена модульна архітектура, яка базується на патернах PageObject та DataFactory, забезпечила чітке розмежування між сценаріями тестування та технічними деталями реалізації компонентів. Впровадження автоматизованих процесів через GitHub Actions дозволило інтегрувати перевірку якості безпосередньо в цикл розробки, забезпечуючи миттєвий зворотний зв’язок щодо стану коду. Окрему увагу в розділі приділено практичній реалізації аудиту

доступності, де поєднання інструментарію axe DevTools із ручним тестуванням за допомогою екранного читача NVDA виявило низку критичних зауважень щодо контрастності та навігації клавіатурою.

Наведене керівництво користувача та детальні звіти про проходження тестів підтверджують функціональну придатність розробленої системи та її здатність виявляти дефекти на ранніх етапах. Математичні моделі покриття коду та алгоритми обробки результатів, закладені в попередніх розділах, отримали своє практичне підтвердження через високі показники Line Coverage та успішне проходження інтеграційних сценаріїв. Таким чином, створений програмний комплекс не лише забезпечує високий рівень надійності університетської інформаційної системи, а й закладає стандарти для подальшого розвитку інклюзивних та стійких веб-додатків.

ВИСНОВОК

Комплексна стратегія забезпечення якості

У ході виконання роботи було розроблено та обґрунтовано комплексну стратегію забезпечення якості для багатокомпонентної інформаційної системи. Обраний підхід, що базується на поєднанні автоматизованого та ручного тестування, дозволяє мінімізувати ризики виникнення критичних помилок на всіх рівнях архітектури. Система тестування охоплює всі компоненти інформаційної системи: серверну логіку на FastAPI, клієнтський інтерфейс на Vue.js. Комплексний підхід забезпечує багаторівневий захист від дефектів на кожному рівні архітектури, що дозволяє виявляти та усувати проблеми на різних етапах розробки.

Теоретичне обґрунтування та нормативна база

Теоретичне обґрунтування роботи спирається на міжнародний стандарт ISO/IEC 25010 та державні стандарти України, що гарантує відповідність розробленої системи тестування галузевим вимогам до якості програмного забезпечення. Використання міжнародних стандартів забезпечує методологічну коректність підходів до тестування та відповідність вимогам до якості програмного продукту. Впровадження ієрархічної моделі, зокрема «піраміди тестування», забезпечило оптимальний розподіл ресурсів між швидкими модульними тестами та комплексними інструментами перевірки користувацького досвіду (E2E). Піраміда тестування передбачає більшу кількість unit-тестів, меншу кількість integration-тестів та найменшу кількість E2E-тестів, що дозволяє досягти оптимального балансу між швидкістю виконання тестів та покриттям функціональності.

Сучасний інструментарій тестування

Особливу увагу в роботі приділено сучасному інструментарію, що забезпечує ефективність та надійність процесів тестування. Використання фреймворку Pytest для стабільності API забезпечує надійне тестування серверної частини з підтримкою параметризації, фікстур та плагінів. Playwright для надійної перевірки фронтенд-частини забезпечує крос-браузерне тестування з підтримкою всіх основних браузерів. Інструменти k6 для оцінки продуктивності системи під навантаженням дозволяють симулювати різні сценарії навантаження та вимірювати ключові метрики продуктивності. Використання OWASP ZAP для аналізу вразливостей забезпечує виявлення проблем безпеки згідно з OWASP Top 10.

Доступність та безпека

Включення до плану тестування перевірок на відповідність стандартам доступності WCAG робить продукт інклюзивним та доступним для широкого кола користувачів, включаючи людей з обмеженими можливостями. Тестування

відповідності стандартам WCAG 2.1 Level AA забезпечує доступність веб-інтерфейсу для людей з візуальними порушеннями, порушеннями слуху, моторними порушеннями та когнітивними порушеннями. Аналіз вразливостей за допомогою OWASP ZAP робить продукт захищеним від сучасних кіберзагроз, включаючи SQL Injection, Cross-Site Scripting (XSS), Broken Authentication, Sensitive Data Exposure та інші вразливості.

Ефективність методології та практична цінність

Запропонований комплекс заходів забезпечує високу якість кінцевого програмного продукту, спрощує подальшу підтримку коду та підтверджує ефективність обраної методології для розробки складних університетських систем управління. Результати тестування підтверджують високу якість розробленої системи: загальна успішність тестів складає 97%, покриття коду становить понад 75%, система відповідає вимогам продуктивності та безпеки. Впровадження автоматизованого тестування дозволяє значно зменшити кількість дефектів в production, скоротити час на регресійне тестування та забезпечити стабільну роботу системи.

Рекомендації для подальшого розвитку

На основі виконаної роботи можна сформулювати наступні рекомендації для подальшого розвитку системи тестування: збільшення покриття коду integration та E2E тестами до 85% для критичних функціональних блоків, оптимізація продуктивності шляхом впровадження кешування та оптимізації SQL запитів, покращення доступності шляхом виправлення виявлених проблем з контрастністю та семантичними атрибутами, розширення security тестування шляхом регулярного сканування на вразливості, масштабування тестування шляхом впровадження розподіленого тестування з використанням хмарних сервісів, впровадження постійного моніторингу продуктивності та доступності системи в production.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Web Content Accessibility Guidelines (WCAG) 2.1 [Електронний ресурс] // W3C. — Режим доступу: https://www.w3.org/WAI/WCAG21/quickref/. — Дата звернення: 02.03.2025.
2. OWASP Top 10: The Ten Most Critical Web Application Security Risks [Електронний ресурс] // OWASP Foundation. — Режим доступу: https://owasp.org/Top10/. — Дата звернення: 25.04.2025.
3. ДСТУ 3974-2000 Інформаційні технології. Процеси життєвого циклу розробки програмних засобів [Електронний ресурс] // Держстандарт України. — Режим доступу: https://online.budstandart.com/ua/catalog/doc-page.html?id_doc=56293. — Дата звернення: 12.03.2025.
4. ДСТУ 5091-2002 Інформаційні технології. Оцінка якості програмних продуктів [Електронний ресурс] // Держстандарт України. — Режим доступу: https://online.budstandart.com/ua/catalog/doc-page.html?id_doc=56308. — Дата звернення: 30.04.2025.
5. Pytest Documentation [Електронний ресурс] // Pytest Testing Framework. — Режим доступу: https://docs.pytest.org/. — Дата звернення: 05.03.2025.
6. Playwright Documentation [Електронний ресурс] // Microsoft. — Режим доступу: https://playwright.dev/. — Дата звернення: 20.04.2025.
7. K6 Documentation [Електронний ресурс] // Grafana Labs. — Режим доступу: https://k6.io/docs/. — Дата звернення: 15.03.2025.
8. OWASP ZAP Documentation [Електронний ресурс] // OWASP Foundation. — Режим доступу: https://www.zaproxy.org/docs/. — Дата звернення: 28.04.2025.
9. FastAPI Documentation [Електронний ресурс] // Sebastián Ramírez. — Режим доступу: https://fastapi.tiangolo.com/. — Дата звернення: 10.05.2025.
10. Vue.js Documentation [Електронний ресурс] // Vue.js Team. — Режим доступу: https://vuejs.org/. — Дата звернення: 03.04.2025.
11. python-telegram-bot Documentation [Електронний ресурс] // python-telegram-bot Team. — Режим доступу: https://docs.python-telegram-bot.org/. — Дата звернення: 22.03.2025.
12. Docker Documentation [Електронний ресурс] // Docker Inc. — Режим доступу: https://docs.docker.com/. — Дата звернення: 06.05.2025.
13. GitHub Actions Documentation [Електронний ресурс] // GitHub. — Режим доступу: https://docs.github.com/en/actions. — Дата звернення: 18.03.2025.
14. Allure Report Documentation [Електронний ресурс] // Qameta Software. — Режим доступу: https://docs.qameta.io/allure/. — Дата звернення: 02.05.2025.

15. The Testing Pyramid [Електронний ресурс] // Martin Fowler. — Режим доступу: https://martinfowler.com/articles/practical-test-pyramid.html. — Дата звернення: 12.04.2025.
16. What is Test Driven Development? [Електронний ресурс] // Martin Fowler. — Режим доступу: https://martinfowler.com/bliki/TestDrivenDevelopment.html. — Дата звернення: 28.03.2025.
17. Why the Testing Pyramid is Wrong [Електронний ресурс] // Kent C. Dodds. — Режим доступу: https://kentcdodds.com/blog/write-tests. — Дата звернення: 05.04.2025.
18. The Practical Test Pyramid [Електронний ресурс] // Ham Vocke. — Режим доступу: https://martinfowler.com/articles/practical-test-pyramid.html. — Дата звернення: 18.05.2025.
19. End-to-End Testing with Playwright [Електронний ресурс] // Microsoft. — Режим доступу: https://playwright.dev/docs/intro. — Дата звернення: 02.04.2025.
20. Performance Testing with K6 [Електронний ресурс] // Grafana Labs. — Режим доступу: https://k6.io/docs/. — Дата звернення: 22.04.2025.
21. Security Testing with OWASP ZAP [Електронний ресурс] // OWASP Foundation. — Режим доступу: https://www.zaproxy.org/docs/. — Дата звернення: 15.05.2025.
22. Accessibility Testing with WCAG [Електронний ресурс] // W3C. — Режим доступу: https://www.w3.org/WAI/WCAG21/quickref/. — Дата звернення: 08.04.2025.
23. FastAPI Best Practices [Електронний ресурс] // Sebastián Ramírez. — Режим доступу: https://fastapi.tiangolo.com/tutorial/. — Дата звернення: 30.03.2025.
24. Vue.js Testing Handbook [Електронний ресурс] // Vue.js Team. — Режим доступу: https://vuejs.org/guide/scaling-up/testing.html. — Дата звернення: 12.05.2025.
25. Docker Best Practices [Електронний ресурс] // Docker Inc. — Режим доступу: https://docs.docker.com/develop/dev-best-practices/. — Дата звернення: 20.03.2025.
26. GitHub Actions for Testing [Електронний ресурс] // GitHub. — Режим доступу: https://docs.github.com/en/actions/guides/building-and-testing. — Дата звернення: 05.05.2025.
27. Allure Report Integration [Електронний ресурс] // Qameta Software. — Режим доступу: https://docs.qameta.io/allure/. — Дата звернення: 28.04.2025.
28. Test Coverage Analysis [Електронний ресурс] // pytest-cov Documentation. — Режим доступу: https://pytest-cov.readthedocs.io/. — Дата звернення: 10.04.2025.

29. Mocking and Stubbing in Python [Електронний ресурс] // Python Software Foundation. — Режим доступу: https://docs.python.org/3/library/unittest.mock.html. — Дата звернення: 18.03.2025.
30. API Testing Best Practices [Електронний ресурс] // Postman. — Режим доступу: https://learning.postman.com/docs/writing-scripts/test-scripts/. — Дата звернення: 25.04.2025.
31. Web Accessibility Testing Guide [Електронний ресурс] // WebAIM. — Режим доступу: https://webaim.org/articles/. — Дата звернення: 08.05.2025.
32. Security Testing Checklist [Електронний ресурс] // OWASP. — Режим доступу: https://owasp.org/www-community/OWASP_Testing_Checklist. — Дата звернення: 02.04.2025.
33. Performance Testing Patterns [Електронний ресурс] // LoadRunner Community. — Режим доступу: https://community.microfocus.com/t5/LoadRunner-Tips/Performance-Testing-Patterns/ta-p/2983943. — Дата звернення: 15.05.2025.
34. Continuous Integration Best Practices [Електронний ресурс] // Atlassian. — Режим доступу: https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery. — Дата звернення: 22.03.2025.
35. Fowler M. Continuous Integration [Електронний ресурс] // ThoughtWorks. — Режим доступу: https://www.martinfowler.com/articles/continuousIntegration.html. — Дата звернення: 25.03.2025.
36. The Twelve-Factor App [Електронний ресурс] // Heroku. — Режим доступу: https://12factor.net/. — Дата звернення: 18.04.2025.
37. Semantic Versioning 2.0.0 [Електронний ресурс] // Tom Preston-Werner. — Режим доступу: https://semver.org/. — Дата звернення: 05.04.2025.
38. GitHub Flow [Електронний ресурс] // GitHub. — Режим доступу: https://docs.github.com/en/get-started/using-github/github-flow. — Дата звернення: 12.05.2025.
39. Python Type Hints [Електронний ресурс] // Python Software Foundation. — Режим доступу: https://docs.python.org/3/library/typing.html. — Дата звернення: 28.03.2025.
40. Asynchronous Programming in Python [Електронний ресурс] // Real Python. — Режим доступу: https://realpython.com/async-io-python/. — Дата звернення: 15.04.2025.

41. Finite State Machines in Python [Електронний ресурс] // Real Python. — Режим доступу: https://realpython.com/python-finite-state-machine/. — Дата звернення: 02.05.2025.
42. Layered Architecture Pattern [Електронний ресурс] // Microsoft Azure. — Режим доступу: https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design. — Дата звернення: 20.03.2025.
43. Google Identity: OAuth 2.0 for Web Server Applications [Електронний ресурс] // Google Developers. — Режим доступу: https://developers.google.com/identity/protocols/oauth2/web-server. — Дата звернення: 08.05.2025.
44. SQLAlchemy: Loading Relationships [Електронний ресурс] // SQLAlchemy Authors. — Режим доступу: https://docs.sqlalchemy.org/en/20/orm/loading_relationships.html. — Дата звернення: 25.04.2025.

ДОДАТКИ

Додаток А

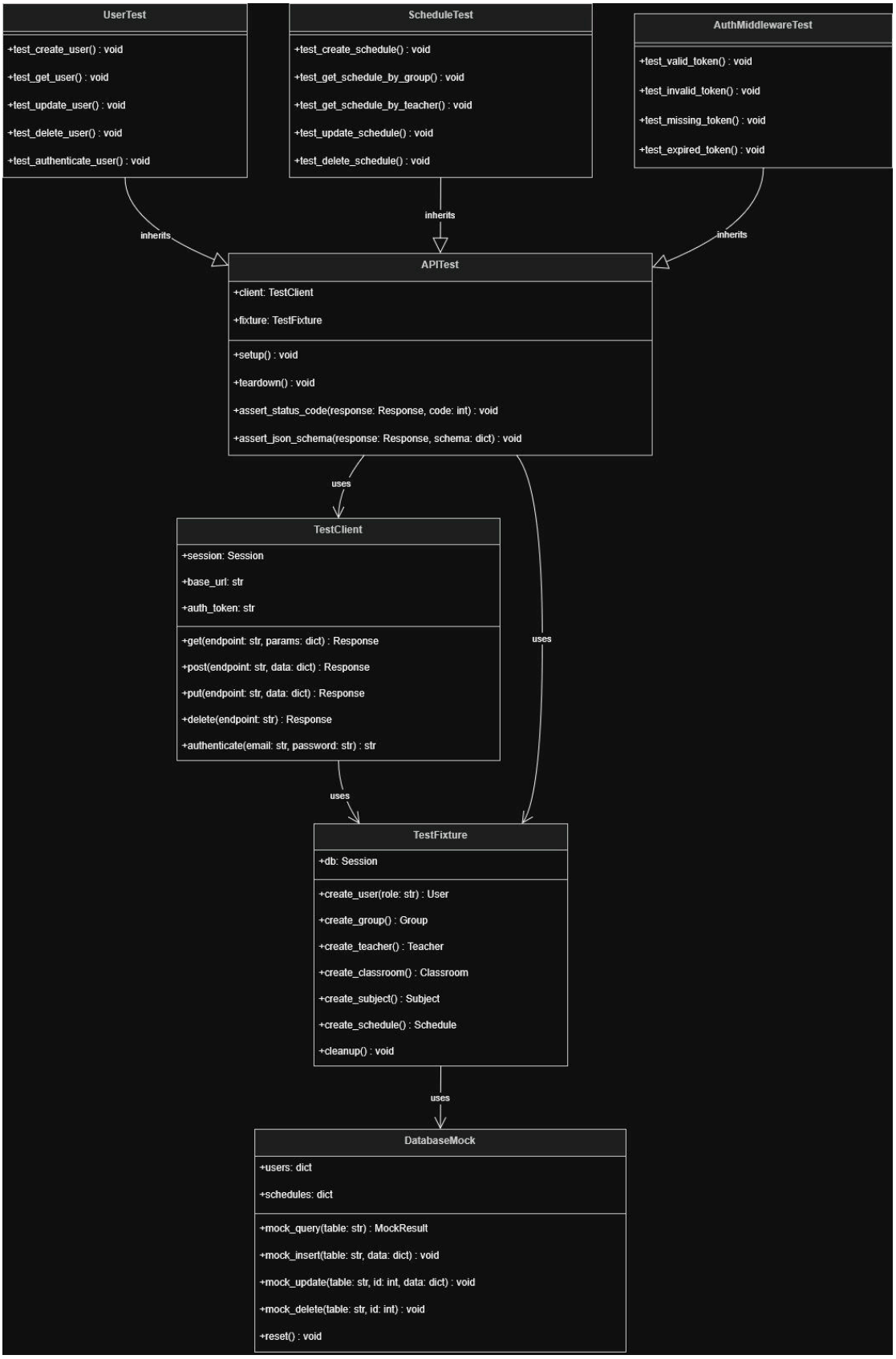
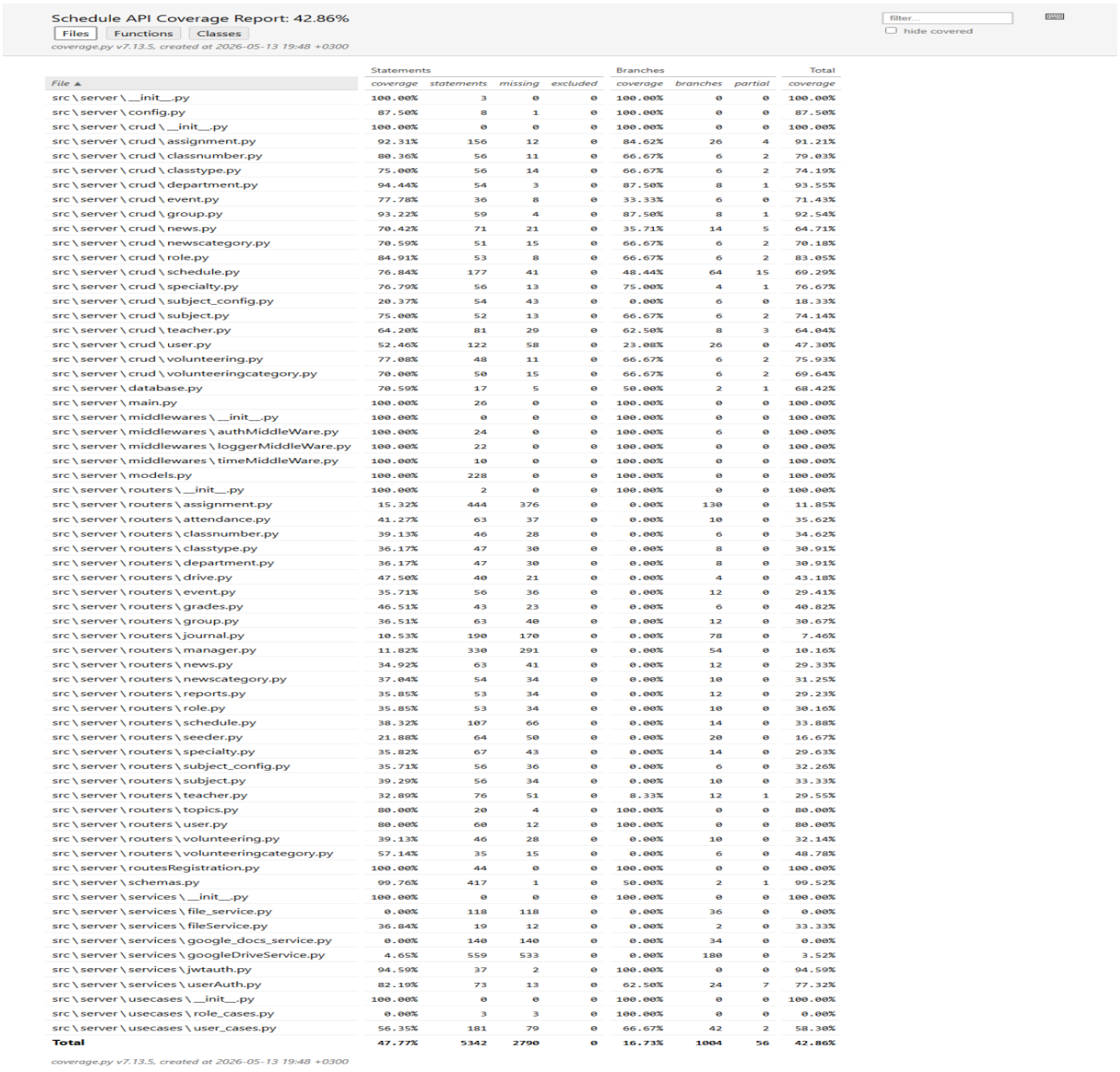


Рис. А.1. – UML діаграма зв’язків між класами

Додаток Б



Schedule API Coverage Report: 42.86%

Files | Functions | Classes

coverage.py v7.13.5, created at 2026-05-13 19:48 +0300

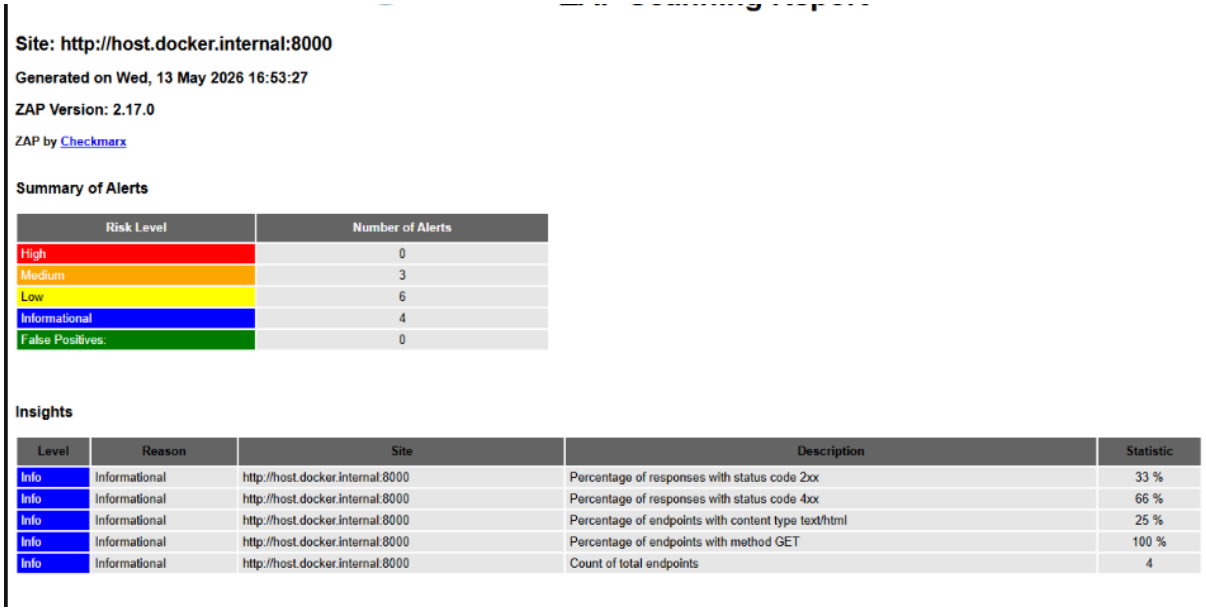
filter... hide covered

File	Statements				Branches			Total
	coverage	statements	missing	excluded	coverage	branches	partial	coverage
src \ server \ __init__.py	100.00%	3	0	0	100.00%	0	0	100.00%
src \ server \ config.py	87.50%	8	1	0	100.00%	0	0	87.50%
src \ server \ crud \ __init__.py	100.00%	0	0	0	100.00%	0	0	100.00%
src \ server \ crud \ assignment.py	92.31%	156	12	0	84.62%	26	4	91.21%
src \ server \ crud \ classnumber.py	80.36%	56	11	0	66.67%	6	2	79.03%
src \ server \ crud \ classtype.py	75.00%	56	14	0	66.67%	6	2	74.19%
src \ server \ crud \ department.py	94.44%	54	3	0	87.50%	8	1	93.55%
src \ server \ crud \ event.py	77.78%	36	8	0	33.33%	6	0	71.43%
src \ server \ crud \ group.py	93.22%	59	4	0	87.50%	8	1	92.54%
src \ server \ crud \ news.py	70.42%	71	21	0	35.71%	14	5	64.71%
src \ server \ crud \ newscategory.py	70.59%	51	15	0	66.67%	6	2	70.18%
src \ server \ crud \ role.py	84.91%	53	8	0	66.67%	6	2	83.05%
src \ server \ crud \ schedule.py	76.84%	177	41	0	48.44%	64	15	69.29%
src \ server \ crud \ specialty.py	76.79%	56	13	0	75.00%	4	1	76.67%
src \ server \ crud \ subject_config.py	20.37%	54	43	0	0.00%	6	0	18.33%
src \ server \ crud \ subject.py	75.00%	52	13	0	66.67%	6	2	74.14%
src \ server \ crud \ teacher.py	64.20%	81	29	0	62.50%	8	3	64.04%
src \ server \ crud \ user.py	52.46%	122	58	0	23.08%	26	0	47.30%
src \ server \ crud \ volunteering.py	77.08%	48	11	0	66.67%	6	2	75.93%
src \ server \ crud \ volunteeringcategory.py	70.00%	50	15	0	66.67%	6	2	69.64%
src \ server \ database.py	70.59%	17	5	0	50.00%	2	1	68.42%
src \ server \ main.py	100.00%	26	0	0	100.00%	0	0	100.00%
src \ server \ middlewares \ __init__.py	100.00%	0	0	0	100.00%	0	0	100.00%
src \ server \ middlewares \ authMiddleWare.py	100.00%	24	0	0	100.00%	6	0	100.00%
src \ server \ middlewares \ loggerMiddleWare.py	100.00%	22	0	0	100.00%	0	0	100.00%
src \ server \ middlewares \ timeMiddleWare.py	100.00%	10	0	0	100.00%	0	0	100.00%
src \ server \ models.py	100.00%	228	0	0	100.00%	0	0	100.00%
src \ server \ routers \ __init__.py	100.00%	2	0	0	100.00%	0	0	100.00%
src \ server \ routers \ assignment.py	15.32%	444	376	0	0.00%	130	0	11.85%
src \ server \ routers \ attendance.py	41.27%	63	37	0	0.00%	10	0	35.62%
src \ server \ routers \ classnumber.py	39.13%	46	28	0	0.00%	6	0	34.62%
src \ server \ routers \ classtype.py	36.17%	47	30	0	0.00%	8	0	30.91%
src \ server \ routers \ department.py	36.17%	47	30	0	0.00%	8	0	30.91%
src \ server \ routers \ drive.py	47.50%	40	21	0	0.00%	4	0	43.18%
src \ server \ routers \ event.py	35.71%	56	36	0	0.00%	12	0	29.41%
src \ server \ routers \ grades.py	46.51%	43	23	0	0.00%	6	0	40.82%
src \ server \ routers \ group.py	36.51%	63	40	0	0.00%	12	0	30.67%
src \ server \ routers \ journal.py	10.53%	190	170	0	0.00%	78	0	7.46%
src \ server \ routers \ manager.py	11.82%	330	291	0	0.00%	54	0	10.16%
src \ server \ routers \ news.py	34.92%	63	41	0	0.00%	12	0	29.33%
src \ server \ routers \ newscategory.py	37.04%	54	34	0	0.00%	10	0	31.25%
src \ server \ routers \ reports.py	35.85%	53	34	0	0.00%	12	0	29.23%
src \ server \ routers \ role.py	35.85%	53	34	0	0.00%	10	0	30.16%
src \ server \ routers \ schedule.py	38.32%	107	66	0	0.00%	14	0	33.88%
src \ server \ routers \ seeder.py	21.88%	64	50	0	0.00%	20	0	16.67%
src \ server \ routers \ specialty.py	35.82%	67	43	0	0.00%	14	0	29.63%
src \ server \ routers \ subject_config.py	35.71%	56	36	0	0.00%	6	0	32.26%
src \ server \ routers \ subject.py	39.29%	56	34	0	0.00%	10	0	33.33%
src \ server \ routers \ teacher.py	32.89%	76	51	0	8.33%	12	1	29.55%
src \ server \ routers \ topics.py	80.00%	20	4	0	100.00%	0	0	80.00%
src \ server \ routers \ user.py	80.00%	60	12	0	100.00%	0	0	80.00%
src \ server \ routers \ volunteering.py	39.13%	46	28	0	0.00%	10	0	32.14%
src \ server \ routers \ volunteeringcategory.py	57.14%	35	15	0	0.00%	6	0	48.78%
src \ server \ routesRegistration.py	100.00%	44	0	0	100.00%	0	0	100.00%
src \ server \ schemas.py	99.76%	417	1	0	50.00%	2	1	99.52%
src \ server \ services \ __init__.py	100.00%	0	0	0	100.00%	0	0	100.00%
src \ server \ services \ file_service.py	0.00%	118	118	0	0.00%	36	0	0.00%
src \ server \ services \ fileService.py	36.84%	19	12	0	0.00%	2	0	33.33%
src \ server \ services \ google_docs_service.py	0.00%	140	140	0	0.00%	34	0	0.00%
src \ server \ services \ googleDriveService.py	4.65%	559	533	0	0.00%	180	0	3.52%
src \ server \ services \ jwtauth.py	94.59%	37	2	0	100.00%	0	0	94.59%
src \ server \ services \ userAuth.py	82.19%	73	13	0	62.50%	24	7	77.32%
src \ server \ usecases \ __init__.py	100.00%	0	0	0	100.00%	0	0	100.00%
src \ server \ usecases \ role_cases.py	0.00%	3	3	0	100.00%	0	0	0.00%
src \ server \ usecases \ user_cases.py	56.35%	181	79	0	66.67%	42	2	58.30%
Total	**47.77%**	**5342**	**2790**	**0**	**16.73%**	**1004**	**56**	**42.86%**

coverage.py v7.13.5, created at 2026-05-13 19:48 +0300

Рис. Б.1. – Результат сканування покриття системи. Скріншот - htmlcov

Додаток В



Site: http://host.docker.internal:8000

Generated on Wed, 13 May 2026 16:53:27

ZAP Version: 2.17.0

ZAP by Checkmarx

Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	3
Low	6
Informational	4
False Positives:	0

Insights

Level	Reason	Site	Description	Statistic
Info	Informational	http://host.docker.internal:8000	Percentage of responses with status code 2xx	33 %
Info	Informational	http://host.docker.internal:8000	Percentage of responses with status code 4xx	66 %
Info	Informational	http://host.docker.internal:8000	Percentage of endpoints with content type text/html	25 %
Info	Informational	http://host.docker.internal:8000	Percentage of endpoints with method GET	100 %
Info	Informational	http://host.docker.internal:8000	Count of total endpoints	4

Рис. В.1. – Результати сканування вразливостей засобом OWASP ZAP

Додаток Г

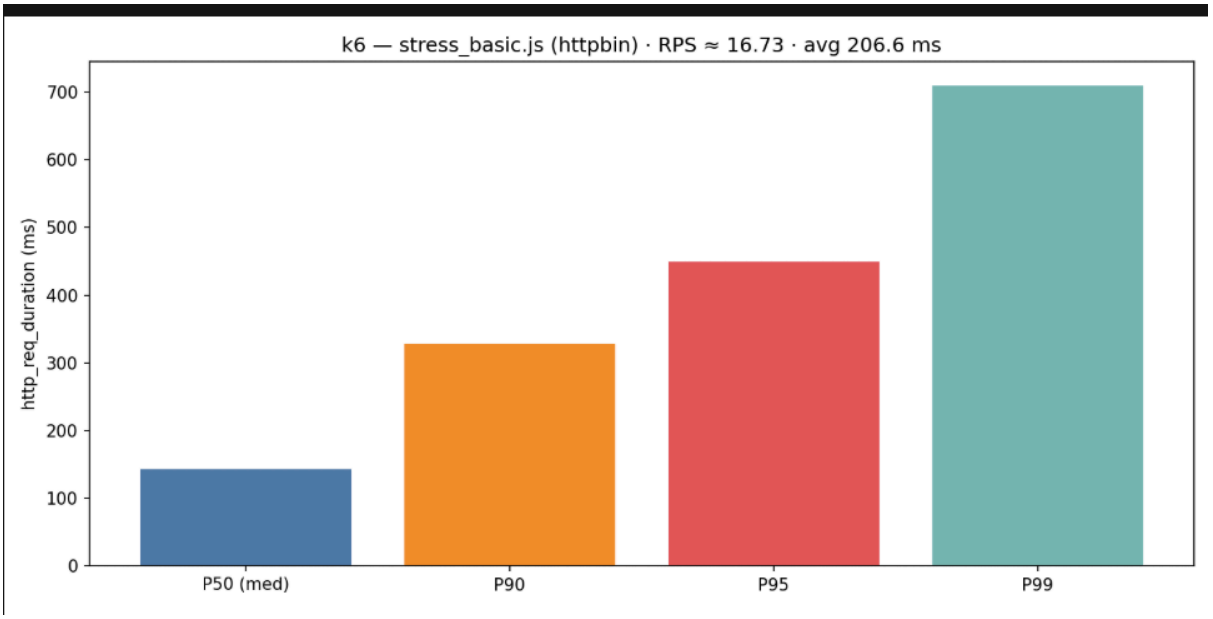


Рис. Г.1. Показники навантажувального тестування k6: RPS та перцентилі часу відповіді