МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління

КАЛЮХ Дмитро Костянтинович

Програмний модуль керування навчальними дефектами інтернет-магазину для формування навичок тестування програмного забезпечення / Software Module for Managing Training Bugs in an Online Store for Developing Software Testing Skills

спеціальність: 122 - Комп'ютерні науки
освітньо-професійна програма - Комп'ютерні науки

Кваліфікаційна робота

Виконав студент групи КН-42
Д. К. Калюх
_____
Науковий керівник
к.т.н., доцент П. Є. Биковий
_____

Кваліфікаційну роботу допущено до захисту
«___»_____ 20___ р.

Завідувач кафедри
_____ Н.В. Дзюбановська

ТЕРНОПІЛЬ – 2026

Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління
Освітній ступінь «бакалавр»
спеціальність: 122 – Комп'ютерні науки
освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Н.В. Дзюбановська
«____» _____20__р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
КАЛЮХУ Дмитру Костянтиновичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи
Програмний модуль керування навчальними дефектами інтернет-магазину для формування навичок тестування програмного забезпечення / Software Module for Managing Training Bugs in an Online Store for Developing Software Testing Skills

керівник роботи к.т.н., доцент, П. Є. Биковий
затверджений наказом по університету від 01 грудня 2025 року № 894

2. Строк подання студентом закінченої кваліфікаційної роботи 25 травня 2026 р.

3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна документація Laravel 12, Bagisto 2.x, офіційна документація до технологій PHP, Vue.js, MySQL.

4. Основні питання, які потрібно розробити:
- Проаналізувати методи та підходи до організації навчальних середовищ для підготовки QA-спеціалістів та огляду існуючих рішень у сфері тестування програмного забезпечення.
- Спроєктувати архітектуру програмного модуля на базі фреймворку Laravel 12 та платформи Bagisto для керування навчальними дефектами інтернет-магазину.
- Розробити систему вмикання/вимикання дефектів через адміністративну панель з поділом на категорії UI/UX, форми та валідація, API/Backend.
- Програмно реалізувати систему, виконати її інтеграцію з платформою Bagisto та провести комплексне тестування розроблених модулів.

5. Перелік графічного матеріалу у роботі:
- Загальна структурна схема програмного модуля керування дефектами.
- UML-діаграма класів пакету BugManager.
- Схема взаємодії компонентів системи (Bagisto + BugManager).
- Діаграма бази даних таблиці bugs.

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 01 грудня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Огляд літературних джерел відповідно до предметної області та складання плану роботи	до 01.01.2026 р.	
2	Написання 1 розділу кваліфікаційної роботи	до 01.02.2026 р.	
3	Написання 2 розділу кваліфікаційної роботи	до 15.03.2026 р.	
4	Написання 3 розділу кваліфікаційної роботи	до 01.05.2026 р.	
5	Представлення попереднього варіанту кваліфікаційної роботи, перевірка та внесення змін керівником	до 15.05.2026 р.	
6	Опрацювання зауважень та представлення завершеного варіанту кваліфікаційної роботи. Підготовка супроводжуючих документів	до 25.05.2026 р.	
7	Перевірка кваліфікаційної роботи на оригінальність тексту	до 30.05.2026 р.	
8	Оформлення кваліфікаційної роботи та отримання допуску до захисту	до 10.06.2026 р.	
9	Подання кваліфікаційної роботи до захисту на засіданні атестаційної комісії	до 10.06.2026 р.	

Студент _____ Д.К. Калюх
підпис

Керівник роботи _____к.т.н., доцент П.Є. Биковий
підпис

АНОТАЦІЯ

Кваліфікаційна робота на тему «Програмний модуль керування навчальними дефектами інтернет-магазину для формування навичок тестування програмного забезпечення» на здобуття освітнього ступеня «бакалавр» зі спеціальності 122 «Комп'ютерні науки» освітньої програми «Комп'ютерні науки» написана обсягом у 56 сторінок і містить 33 ілюстрацій, 7 таблиць, 2 додатки та 29 використаних джерел.

Метою кваліфікаційної роботи є розробка та програмна реалізація програмного модуля керування навчальними дефектами на базі платформи Bagisto та фреймворку Laravel 12, що забезпечує формування практичних навичок тестування програмного забезпечення у QA-спеціалістів.

Методами дослідження є методи системного аналізу, об'єктно-орієнтованого проєктування, методи розробки веб-застосунків на основі патерну MVC, а також методи тестування програмного забезпечення.

Внаслідок виконання роботи розроблено програмний модуль BugManager у вигляді пакету для платформи Bagisto, який дозволяє через адміністративну панель вмикати та вимикати навчальні дефекти інтернет-магазину. Реалізовано дев'ять типів дефектів трьох категорій: UI/UX, форми та валідація, API/Backend. Система побудована на базі Laravel 12 з використанням ServiceProvider, Middleware та Eloquent ORM. Керування дефектами здійснюється через зручний веб-інтерфейс з відображенням статусу кожного дефекту та можливістю миттєвого вмикання і вимикання через toggle-перемикачі.

Розроблений програмний продукт є готовим до використання навчальним середовищем, яке може бути розгорнуте як самостійний сервіс для підготовки QA-спеціалістів або інтегроване в навчальні програми з тестування програмного забезпечення.

Ключові слова: ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, QA, ДЕФЕКТИ, LARAVEL, BAGISTO, ІНТЕРНЕТ-МАГАЗИН, MIDDLEWARE, SERVICEPROVIDER, WEB-ЗАСТОСУНОК.

ANNOTATION

Qualification work on the topic «Software Module for Managing Training Bugs in an Online Store for Developing Software Testing Skills» for obtaining a bachelor's degree in specialty 122 «Computer Science» of the educational program «Computer Science» is written on 56 pages and contains 33 illustrations, 7 tables, 2 attachments, and 29 references.

The aim of the qualification work is to develop and programmatically implement a software module for managing training bugs based on the Bagisto platform and Laravel 12 framework, which ensures the development of practical software testing skills for QA specialists.

Research methods include system analysis, object-oriented design, web application development methods based on the MVC pattern, and software testing methods.

As a result of the work, the BugManager software module was developed as a package for the Bagisto platform, which allows enabling and disabling training bugs of an online store through an administrative panel. Nine types of bugs in three categories were implemented: UI/UX, forms and validation, API/Backend. The system is built on Laravel 12 using ServiceProvider, Middleware, and Eloquent ORM. Bug management is carried out through a user-friendly web interface displaying the status of each bug with the ability to instantly enable and disable them via toggle switches.

The developed software product is a ready-to-use training environment that can be deployed as a standalone service for training QA specialists or integrated into software testing educational programs.

Keywords: SOFTWARE TESTING, QA, BUGS, LARAVEL, BAGISTO, ONLINE STORE, MIDDLEWARE, SERVICEPROVIDER, WEB APPLICATION.

ЗМІСТ

Вступ 7
1 Аналіз предметної області та постановка задачі 9
1.1 Аналіз предметної області 9
1.2 Огляд та аналіз існуючих рішень 11
1.3 Постановка задачі та вимоги до програмного модуля 18
2 Алгоритмічне та інформаційне забезпечення модуля 22
2.1 Загальна архітектура системи 22
2.2 Алгоритм роботи модуля BugManager 24
2.3 Інформаційне забезпечення системи 25
3 Програмно-технологічне забезпечення модуля 28
3.1 Реалізація програмного забезпечення 28
3.2 Реалізація механізмів ін'єкції дефектів 33
3.3 Інтерфейс адміністративної панелі 34
3.4 Аналіз роботи реалізованих дефектів 36
3.5 Тестування розробленого модуля 41
Висновки 44
Список використаних джерел 46
Додаток А Копія публікації 48
Додаток Б Лістинг вихідного коду програмного модуля BugManager 52

ВСТУП

Стрімкий розвиток інформаційних технологій та цифровізація бізнес-процесів зумовили значне зростання попиту на кваліфікованих спеціалістів із тестування програмного забезпечення. Якість програмних продуктів безпосередньо залежить від рівня підготовки QA-інженерів, які здатні своєчасно виявляти та документувати дефекти у складних системах. Однак підготовка таких фахівців стикається з суттєвою проблемою: для формування практичних навичок тестування необхідне реальне середовище з керованими дефектами, яке дозволяє відтворювати різноманітні помилки у контрольованих умовах.

Сучасні підходи до навчання QA-спеціалістів переважно базуються на теоретичних матеріалах або на тестуванні готових застосунків із заздалегідь відомими помилками. Такі підходи мають суттєві недоліки: теоретичні знання без практики не формують стійких навичок, а статичні тестові середовища не відображають реальних умов роботи. Відповідно, виникає потреба у динамічному навчальному середовищі, де інструктор може оперативно вмикати та вимикати різні типи дефектів залежно від навчальної програми.

Метою дослідження є розробка та програмна реалізація програмного модуля керування навчальними дефектами інтернет-магазину на базі платформи Bagisto та фреймворку Laravel 12 для формування практичних навичок тестування програмного забезпечення.

Завдання дослідження:

1) проаналізувати предметну область, методи підготовки QA-спеціалістів та існуючі підходи до організації навчальних середовищ для тестування програмного забезпечення;

2) спроєктувати архітектуру програмного модуля BugManager у вигляді пакету для платформи Bagisto з використанням Laravel 12;

3) розробити систему керування дефектами з поділом на категорії UI/UX, форми та валідація, API/Backend із можливістю оперативного вмикання та вимикання через адміністративну панель;

4) програмно реалізувати модуль з використанням об'єктно-орієнтованого підходу, мови PHP, фреймворку Laravel 12, платформи Bagisto та провести комплексне тестування розроблених компонентів.

Об'єктом дослідження є процес формування практичних навичок тестування програмного забезпечення у QA-спеціалістів в умовах контрольованого навчального середовища.

Предметом дослідження є архітектурні та програмно-технологічні рішення для реалізації системи керування навчальними дефектами інтернет-магазину на базі фреймворку Laravel 12 та платформи Bagisto.

Розроблений програмний модуль є готовим до використання навчальним інструментом, який може бути розгорнутий як самостійний сервіс або інтегрований у існуючі навчальні програми підготовки QA-спеціалістів. Запропоновані архітектурні рішення на базі патерну ServiceProvider та Middleware дозволяють легко розширювати набір навчальних дефектів без зміни основного коду платформи, що забезпечує гнучкість та масштабованість системи.

Основні результати кваліфікаційної роботи апробовано на IV Всеукраїнській науково-практичній конференції студентів, аспірантів та молодих вчених «Інтелектуальні комп’ютерні системи та мережі» (м. Тернопіль, 20 травня 2026 р.), де вони були представлені та опубліковані у вигляді тез «Програмний модуль керування навчальними дефектами інтернет-магазину для формування навичок тестування програмного забезпечення» (Додаток А).

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної області

У сучасну цифрову епоху, яка характеризується безпрецедентними темпами глобалізації та технологічного розвитку, якість програмного забезпечення остаточно утвердилася як один із фундаментальних чинників успіху та конкурентоспроможності будь-якого технологічного продукту. Особливої критичної ваги ця інженерна парадигма набуває у сфері електронної комерції, де цифрова платформа є не просто вітриною, а єдиною точкою генерації прибутку. Будь-яка технічна несправність у таких системах — від мілісекундної затримки завантаження сторінки до критичної помилки в алгоритмах розрахунку вартості у кошику — має миттєвий та безпосередній вплив на рівень конверсії та фінансові показники бізнесу.

Статистичні дослідження демонструють, що глобальний ринок електронної комерції невпинно зростає. У 2026 році глобальний ринок досягне позначки у 5,31 трлн доларів США [4]. В Україні цей тренд також має яскраво виражений характер: вітчизняний e-commerce виходить із фази «післяпандемічного хайпу» і переходить до етапу стабільного розвитку, де онлайн-частка у роздрібній торгівлі зростає на 17-25% щорічно [7].

Водночас наслідки програмних дефектів стають дедалі катастрофічнішими для корпоративного сектору. Аналітичні дані консорціумів з якості ПЗ свідчать, що низька якість програмного забезпечення коштує компаніям у Сполучених Штатах понад 2,08 трильйона доларів щорічно [23]. У контексті систем електронної комерції наслідки дефектів є надзвичайно гострими: єдиний дефект у процесі оформлення замовлення може призвести до показника покинутих кошиків на рівні понад 75%. Поведінкові метрики сучасних користувачів демонструють абсолютну нульову толерантність до технічних недоліків: 68% споживачів назавжди покидають додаток після того, як стикаються лише з двома програмними помилками [23]. Більше того, технічний борг та постійне виправлення дефектів поглинають від 30% до 50% робочого часу команд розробки.

Саме тому превентивне виявлення дефектів та висококваліфікована підготовка фахівців із забезпечення якості стають критично важливими бізнес-імперативами. Ця тенденція підкріплюється потужним зростанням українського IT-сектору: у 2025 році український IT-експорт повернувся до зростання і сягнув 6,66 млрд доларів США [10]. Грудень 2025 року став одним із найуспішніших місяців за останні п'ять років, принісши 685 млн доларів, що свідчить про високу довіру іноземних партнерів до українських інженерів. Потреба у кваліфікованих спеціалістах стрімко зростає: лише у січні 2025 року IT-компанії опублікували майже 11 000 нових вакансій [8].

Тренди інженерії якості на 2026 рік диктують безальтернативний перехід від простого ручного виконання тестових сценаріїв до побудови систем спостережуваності, впровадження «самовідновлюваної» автоматизації та використання автономних AI-агентів, здатних самостійно сканувати UI на предмет змін та оновлювати селектори [5; 21]. Від сучасного QA-інженера вимагається не просто вміння писати скрипти, а здатність виступати архітектором якості та глибоко розуміти логіку складних мікросервісних систем.

Предметна область поточного дослідження поєднує дві площини: функціонування інтернет-магазину як комплексної інформаційної системи (каталог товарів, пошук, кошик, оформлення замовлення, профіль користувача) та організацію навчального процесу з тестування ПЗ, де дефекти є «навчальними кейсами». Система електронної комерції як об'єкт тестування є особливо цінною для навчання через свою комплексність: вона включає одночасно UI/UX компоненти, складну бізнес-логіку розрахунків, API-інтеграції та механізми авторизації. Це дозволяє навчати QA-спеціалістів виявляти дефекти одразу в усіх ключових категоріях.

У навчальному контексті «дефект» трактується як відхилення фактичної поведінки системи від очікуваної згідно зі специфікацією. Організаційною формою управління дефектами є імітація реального трекера (подібного до Jira), де баги проходять чіткий життєвий цикл: new → triaged → confirmed → fixed → closed. Такий підхід формує у студентів не лише технічні навички виявлення дефектів, але й розуміння процесів управління якістю в реальних IT-командах.

Проте підготовка QA-інженерів стикається з фундаментальною педагогічною проблемою: відсутністю реалістичних, масштабованих та безпечних навчальних середовищ. Традиційні академічні підходи часто обмежуються аналізом статичних фрагментів коду, що абсолютно не відображає динаміку «живої» виробничої системи e-commerce, де дефекти рідко бувають ізольованими і часто мають складний каскадний характер. Якщо дефекти у навчальному стенді є статичними — студенти швидко обмінюються розв'язаннями; якщо ж вони вигадуються на ходу — втрачається відтворюваність та можливість об'єктивного оцінювання.

Актуальність даної роботи обґрунтована гострою потребою в оптимізації процесу навчання спеціалістів із тестування ПЗ шляхом створення спеціалізованого освітнього модуля «контрольованої дефектності». Концепція ін'єкції дефектів давно визнана в системній інженерії як найефективніший метод підвищення покриття тестами та валідації стійкості систем до відмов [14]. Застосування цього підходу в освітніх цілях дозволяє перетворити еталонну платформу інтернет-магазину на керовану лабораторію дефектів, де адміністратор (викладач) формує каталог навмисно закладених багів та активує їх для конкретних навчальних груп.

1.2 Огляд та аналіз існуючих рішень

Для обґрунтування вибору архітектурних рішень необхідно здійснити аналіз існуючих підходів до організації навчальних середовищ для тестування програмного забезпечення. У наявних підходах можна виокремити декілька класів рішень, кожен з яких частково закриває потреби навчальної лабораторії, але має певні обмеження.

Першим класом є спеціально розгорнуті тестові інтернет-магазини зі статичними помилками. Наприклад, для освітніх цілей використовувався вебсайт prestashop.qatestlab.com.ua, де студенти могли практикуватися у пошуку типових дефектів електронної комерції. Серед демонстрованих помилок були: дефекти при

масштабуванні сайту, що порушували адаптивність інтерфейсу, дефекти бізнес-логіки розрахунку знижок та функціональні дефекти інтеграції доставки.

Перевагою таких стендів є реалістичне середовище e-commerce та різноманітність типів дефектів. Однак фундаментальним недоліком є статичність: оскільки дефекти жорстко закодовані, студенти швидко запам'ятовують їх розташування або обмінюються відповідями, що зводить нанівець педагогічну цінність інструменту в довгостроковій перспективі. Крім того, такі стенди неможливо адаптувати під конкретну навчальну програму або рівень підготовки групи.

Зовнішній вигляд навчального стенду PrestaShop QATestLab наведено на рисунку 1.1. На рисунку 1.2 показано приклад дефекту бізнес-логіки розрахунку знижки, що був доступний для практики на цьому стенді.

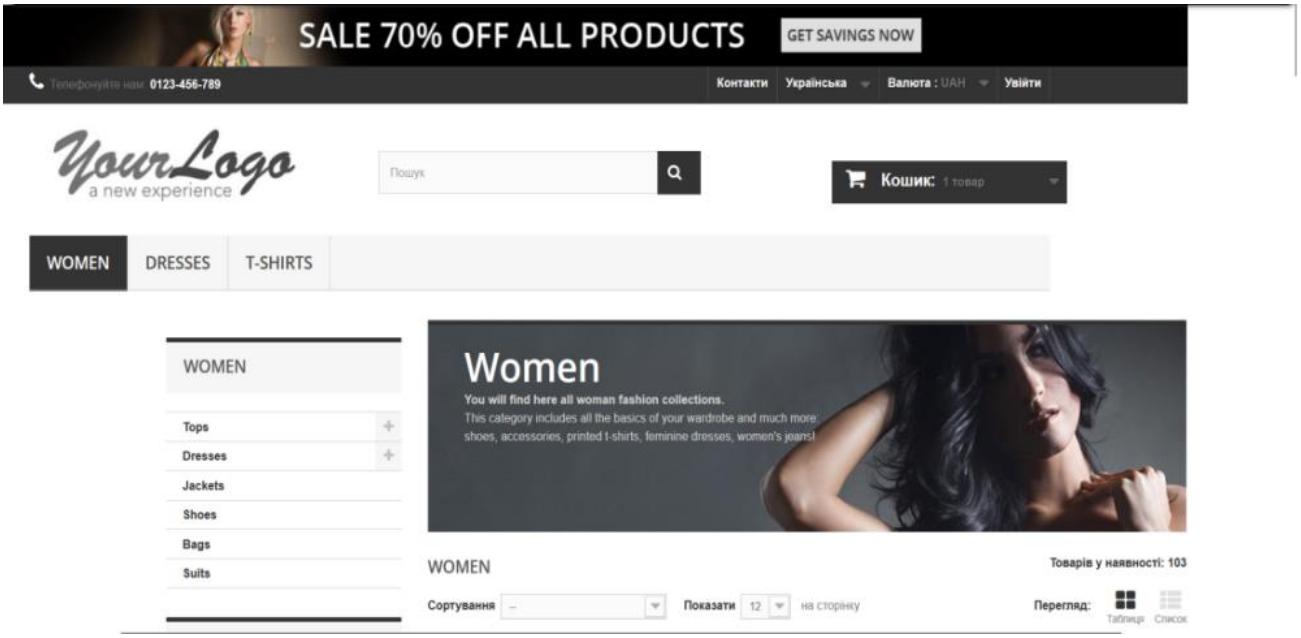

Рисунок 1.1 — Головна сторінка навчального стенду PrestaShop QATestLab

Другим класом є готові навчальні застосунки для тестування безпеки — Vulnerable Web Applications . Найвідомішими представниками є OWASP Juice Shop, WebGoat, bWAPP та DVWA [25]. OWASP Juice Shop є сучасним SPA-додатком на Node.js та Angular, який містить понад 100 завдань різного рівня складності. Він широко використовується у навчанні пентестерів та security-інженерів. Проте для підготовки QA-спеціалістів широкого профілю він має суттєві обмеження: фокусується виключно на security-вразливостях відповідно до

OWASP Top 10, ігноруючи звичайні бізнес-логічні чи UI дефекти класичного e-commerce. Крім того, він не є реальним інтернет-магазином з повноцінним каталогом товарів, кошиком та процесом оформлення замовлення. WebGoat від OWASP орієнтований суто на навчання веб-безпеці та не підходить для навчання функціональному тестуванню.

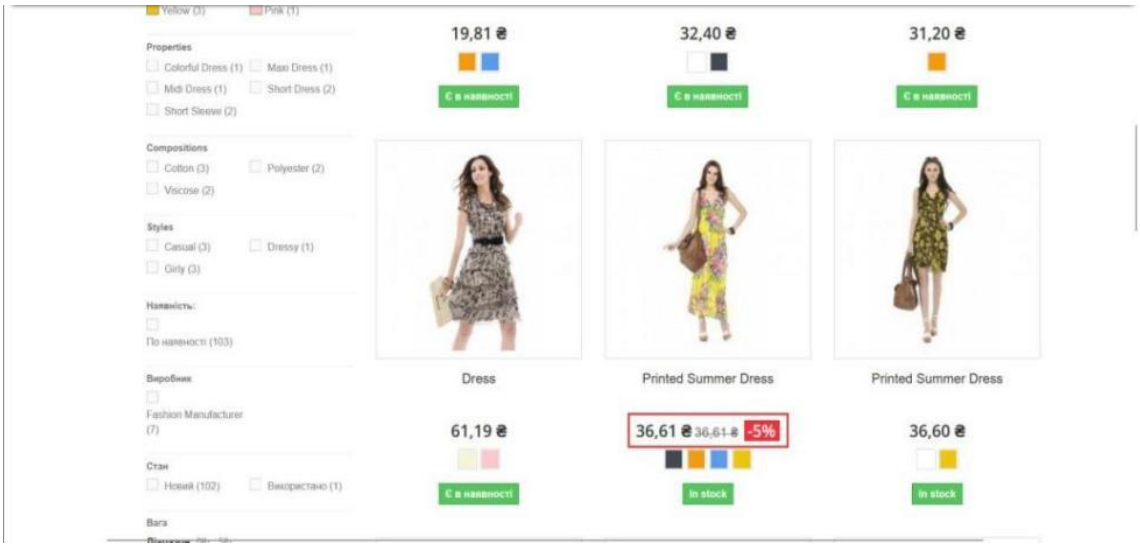

Рисунок 1.2 – Приклад дефекту розрахунку знижки на стенді PrestaShop QATestLab

Інтерфейс OWASP Juice Shop наведено на рисунку 1.3. Як видно з рисунку, система є сучасним SPA-додатком, однак не відтворює реального середовища інтернет-магазину з повноцінним каталогом товарів та процесом оформлення замовлення.

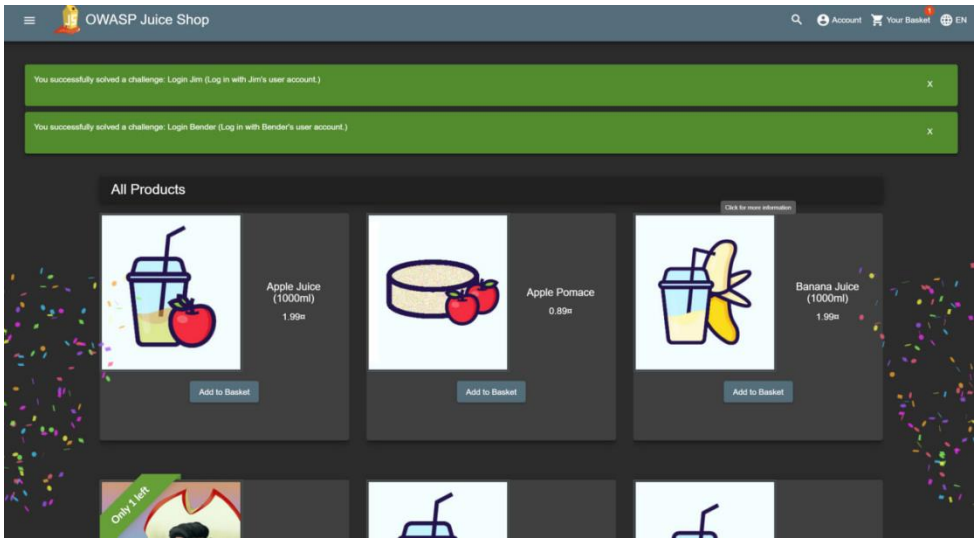

Рисунок 1.3 – Головна сторінка OWASP Juice Shop

Додатки на кшталт bWAPP та DVWA статично поламані на рівні жорстко закодованих помилок, що позбавляє викладача можливості динамічно керувати навчальними сценаріями та адаптувати їх під різні групи студентів.

Сторінку входу до застосунку DVWA наведено на рисунку 1.4. Система розгортається локально та надає набір навмисно вразливих веб-сторінок для відпрацювання навичок пентестування. Однак відсутність повноцінного e-commerce середовища та неможливість динамічного керування дефектами обмежують її застосування для навчання QA-спеціалістів широкого профілю.

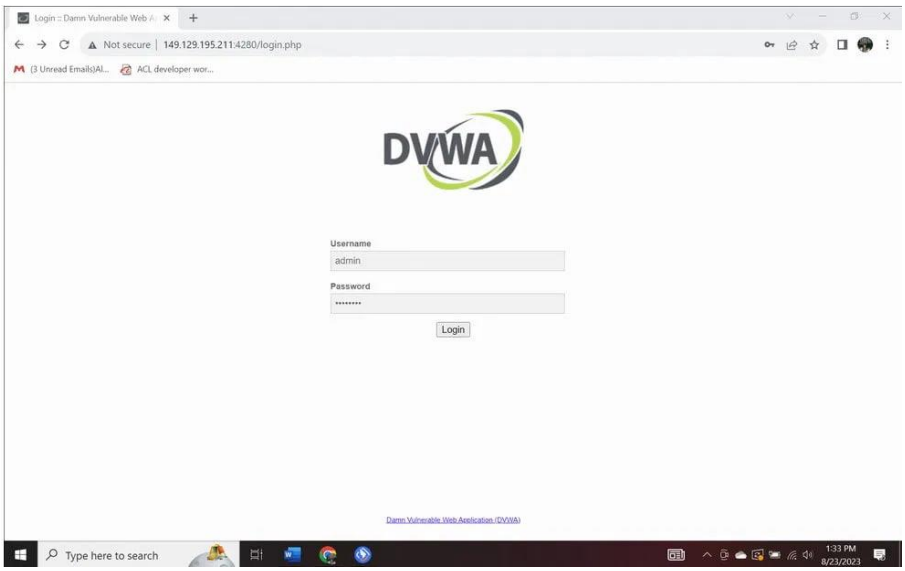

Рисунок 1.4 – Сторінка входу до застосунку DVWA

Інтерфейс застосунку bWAPP наведено на рисунку 1.5. Як видно з рисунку, система надає список із понад 100 типів вразливостей (HTML Injection, LDAP Injection тощо), однак усі вони є статично закодованими та орієнтованими виключно на security-тестування, що не відповідає потребам навчання функціональному тестуванню e-commerce систем.

Третім класом є класичні системи управління дефектами — баг-трекери. Jira Software від Atlassian та GitHub Issues є найпоширенішими інструментами управління дефектами в індустрії. Вони ефективно організовують реєстрацію, тріаж та відстеження статусу дефектів, надають зручний інтерфейс для командної роботи та інтегруються з системами контролю версій. Однак вони є виключно організаційними інструментами і не забезпечують «ін'єкцію» дефектів у сам

продукт. Використання баг-трекерів у навчанні доцільне як доповнення до навчального стенду, але не як самостійний засіб підготовки QA-спеціалістів.

Рисунок 1.5 – Інтерфейс навчального застосунку bWAPP з переліком вразливостей для тестування

Інтерфейс системи управління дефектами Jira Software наведено на рисунку 1.6. Як видно з рисунку, система надає зручний інтерфейс для відстеження статусу дефектів та організації командної роботи, однак не має жодних механізмів для впровадження дефектів у веб-застосунок.

Четвертим класом є готові e-commerce платформи на Laravel. Серед них виділяються Bagisto, Aimeos та InnoShop. Ці платформи надають повноцінне середовище інтернет-магазину, але в базовій конфігурації не мають механізмів динамічного керування дефектами. Вони потребують спеціальної надбудови у вигляді окремого модуля, що і є предметом даної кваліфікаційної роботи.

Для наочного порівняння адміністративних панелей розглянуто три платформи. Адміністративна панель Bagisto наведена на рисунку 1.7 — вона надає повноцінний інтерфейс керування магазином, каталогом товарів,

замовленнями та клієнтами, однак не містить жодних інструментів для керування навчальними дефектами у базовій конфігурації.

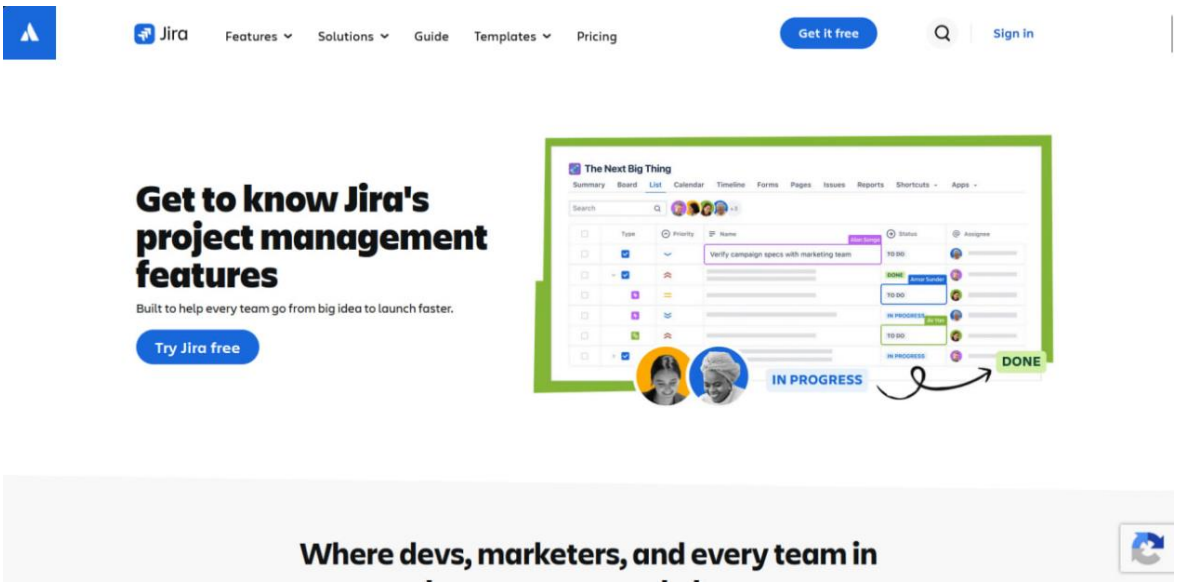

Рисунок 1.6 – Інтерфейс системи управління дефектами Jira Software

П'ятим класом є хмарні навчальні платформи для тестування, такі як Sauce Labs Training, Testim Academy та різноманітні онлайн-курси на Udemy і Coursera. Вони надають навчальні матеріали та симульовані середовища для тестування, але мають суттєвий недолік — відсутність можливості кастомізації під конкретну навчальну програму та необхідність постійного підключення до інтернету.

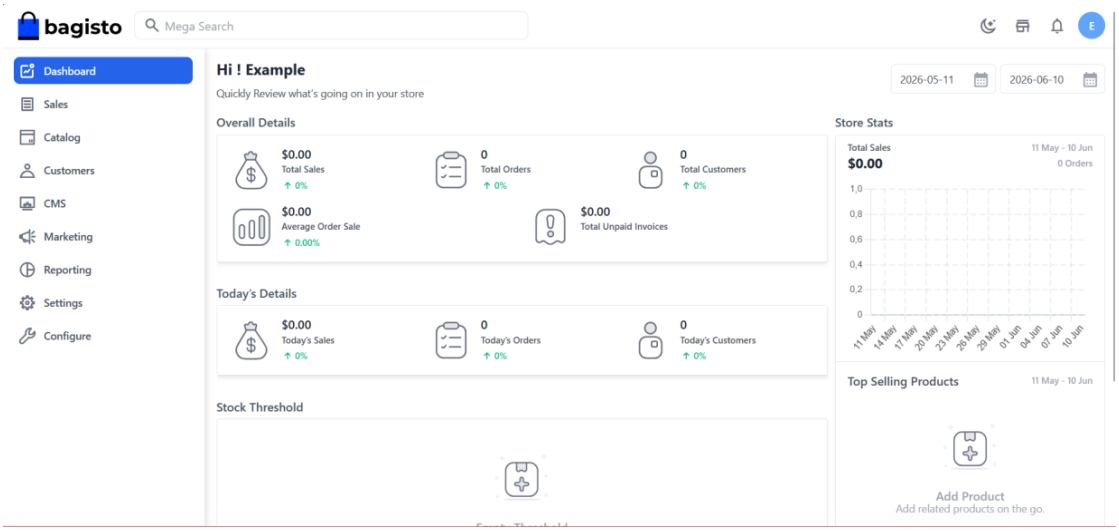

Рисунок 1.7 – Адміністративна панель платформи Bagisto у базовій конфігурації

Окрім зазначених недоліків, хмарні платформи зазвичай надають обмежений набір типів дефектів, що не відображає повного спектру помилок реальних e-commerce систем. Вартість підписки на такі платформи також є суттєвим обмеженням для навчальних закладів. Наприклад, Sauce Labs пропонує навчальні матеріали переважно для автоматизованого тестування, залишаючи поза увагою ручне функціональне тестування бізнес-логіки інтернет-магазинів. Інтерфейс платформи Sauce Labs наведено на рисунку 1.8.

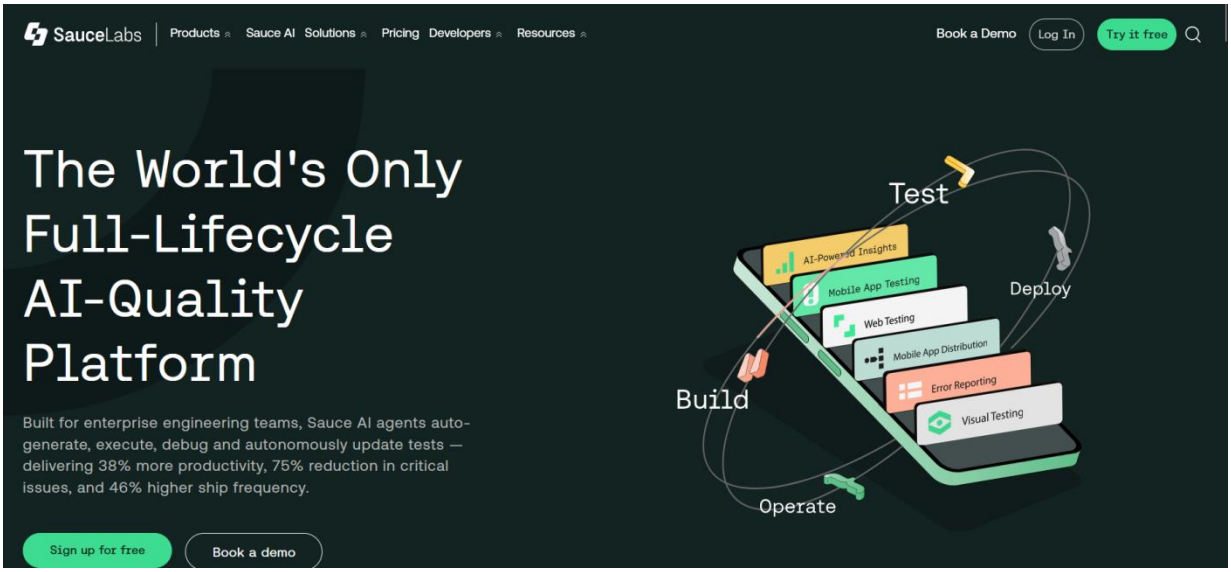

Рисунок 1.8 – Інтерфейс хмарної платформи для тестування Sauce Labs

Порівняльну характеристику існуючих рішень наведено у таблиці 1.1. Показано, що жодне з існуючих рішень не забезпечує динамічного керування дефектами різних категорій та автономності розгортання. Це підтверджує актуальність та доцільність розробки запропонованого програмного модуля.

Аналіз існуючих рішень дозволяє сформулювати ключові вимоги до розроблюваної системи, яких бракує наявним інструментам: динамічне вмикання та вимикання дефектів без перезапуску системи; поділ дефектів на категорії відповідно до реальних типів помилок у e-commerce (UI/UX, форми та валідація, API/Backend); збереження стану дефектів у базі даних для забезпечення відтворюваності навчальних сценаріїв; зручний адміністративний інтерфейс для викладача; незалежність модуля від основного коду платформи.

Таблиця 1.1 – Порівняльна характеристика існуючих рішень для навчання тестуванню

Рішення	Тип дефектів	Динамічне керування	Платформа e-commerce	Автономність	Придатність
PrestaShop QATestLab	UI, функціональні	Ні	Так	Так	Обмежена
OWASP Juice Shop	Security	Ні	Частково	Так	Часткова
bWAPP / DVWA	Security	Ні	Ні	Так	Часткова
WebGoat	Security	Ні	Ні	Так	Часткова
Jira / GitHub Issues	—	—	Ні	Так	Організаційна
Хмарні платформи	Різні	Обмежено	Частково	Ні	Часткова
Запропонований модуль	UI, форми, API	Так	Так (Bagisto)	Так	Повна

1.3 Постановка задачі та вимоги до програмного модуля

Для забезпечення наукової обґрунтованості підходу до розробки навчального модуля методологія базується на новітньому національному стандарті якості ПЗ — ДСТУ ISO/IEC 25010:2025 «Інженерія систем і програмних засобів. Вимоги до якості систем і програмних засобів та її оцінювання (SQuaRE). Модель якості продукту» [3]. Цей стандарт визначає вісім головних характеристик якості, критичних для e-commerce систем, які наведено у таблиці 1.2.

Стандарт ДСТУ ISO/IEC 25010:2025 є прямою заміною попередньої версії ISO/IEC 25010:2011 і враховує сучасні реалії розробки програмного забезпечення, зокрема хмарні технології, мікросервісну архітектуру та вимоги до безпеки відповідно до актуальних загроз. Використання цього стандарту як методологічної основи забезпечує відповідність розробленого модуля міжнародним вимогам до якості програмного забезпечення.

Таблиця 1.2 – Характеристики якості ПЗ відповідно до ДСТУ ISO/IEC 25010:2025

Характеристика	Опис	Застосування у навчальному модулі
Функціональність	Ступінь забезпечення функцій, що відповідають потребам	Перевірка коректності функцій кошика та замовлень
Надійність	Здатність виконувати функції за визначених умов	Симуляція збоїв для навчання перевірці відновлюваності
Зручність використання	Ступінь ефективності та задоволеності користувача	Впровадження UI дефектів для виявлення порушень
Ефективність продуктивності	Продуктивність відносно використаних ресурсів	Симуляція деградації часу відгуку
Сумісність	Здатність обмінюватися інформацією	Перевірка інтеграції API компонентів
Безпека	Захист інформації та даних	Тренування виявлення вразливостей
Супроводжуваність	Ефективність модифікації та тестування	Оцінка тестованості компонентів
Переносимість	Адаптивність до різних середовищ	Перевірка адаптивності інтерфейсу

Для об'єктивної оцінки якості тестування використовуються стандартизовані метрики якості програмного забезпечення. Впровадження цих вимірюваних показників в освітній процес формує аналітичне інженерне мислення у студентів [22]. Ключовими метриками є наступні:

Щільність дефектів — відношення кількості підтверджених дефектів (багів) до загального обсягу коду. Ця метрика розраховується за формулою:

$$DD = \frac{D_{total}}{LOC \times 1000}, \quad (1.1)$$

де D_{total} — кількість знайдених дефектів;

LOC — кількість рядків коду.

Метрика дозволяє об'єктивно ідентифікувати проблемні модулі в архітектурі системи та порівнювати якість різних компонентів між собою.

Витік дефектів — відсоток багів, які не були знайдені QA-командою на тестових середовищах і потрапили у виробниче середовище. Формула розрахунку:

$$DL = \frac{D_{prod}}{D_{qa}+D_{prod}} \times 100, \quad (1.2)$$

де D_{prod} — дефекти, знайдені після релізу,

а D_{qa} — дефекти, виявлені до релізу. Це головний індикатор ефективності стратегії тестування.

Відсоток виявлення дефектів — оцінює ефективність набору тест-кейсів:

$$EDR = \frac{D_{total\_found}}{T_{executed}} \times 100, \quad (1.3)$$

де $T_{executed}$ — загальна кількість виконаних тестових сценаріїв.

Ефективність усунення дефектів — дозволяє оцінити, наскільки успішно команда розробки справляється з усуненням знайдених багів на кожному етапі життєвого циклу розробки:

$$DRE = \frac{D_{removed}}{D_{removed}+D_{remaining}} \times 100, \quad (1.4)$$

де $D_{removed}$ — кількість усунених дефектів,

$D_{remaining}$ — кількість дефектів що залишились.

Висока ефективність усунення дефектів на ранніх етапах розробки свідчить про зрілість процесів забезпечення якості в команді.

Розроблений програмний модуль керування навчальними дефектами створюється для того, щоб дозволити викладачеві штучно маніпулювати цими метриками у навчальному середовищі. Викладач може свідомо активувати певну кількість логічних дефектів різних категорій, щоб емпірично оцінити, який відсоток студент зможе виявити самостійно за відведений час. Порівнюючи кількість активованих дефектів з кількістю знайдених, викладач отримує об'єктивну метрику EDR для кожного студента, яка відображає реальний рівень його підготовки.

Важливим методологічним принципом є класифікація навчальних дефектів за категоріями відповідно до реальних типів помилок у системах e-commerce. На підставі аналізу галузевих звітів про дефекти в системах електронної комерції та вимог стандарту ДСТУ ISO/IEC 25010:2025 виокремлено три основні категорії дефектів для навчального модуля, наведені у таблиці 1.3.

На підставі проведеного аналізу предметної області, існуючих рішень та методологічного обґрунтування сформульовано наступні вимоги до розроблюваної системи: наявність повноцінної e-commerce платформи як основи для тестування; можливість динамічного вмикання та вимикання дефектів через адміністративну панель без перезапуску системи; поділ дефектів на категорії UI/UX, форми та валідація, API/Backend; збереження стану дефектів у реляційній базі даних для забезпечення відтворюваності; відображення severity (рівня критичності) кожного дефекту; незалежність модуля від основного коду платформи для забезпечення можливості оновлення; зручний адміністративний інтерфейс з можливістю групового вимкнення всіх дефектів.

Таблиця 1.3 – Класифікація навчальних дефектів за категоріями

Категорія	Опис	Приклади дефектів	Характеристика якості
UI/UX	Дефекти інтерфейсу користувача	Дублювання товарів, зламаний лічильник кошика	Зручність використання
Форми та валідація	Дефекти перевірки вхідних даних	Відсутня валідація email, від'ємна кількість	Функціональність, Безпека
API/Backend	Дефекти серверної логіки та API	Витік даних, неправильний HTTP статус	Безпека, Надійність

Для реалізації цих вимог обрано платформу Bagisto 2.x на базі фреймворку Laravel 12, що забезпечує готовий storefront та повноцінний адміністративний бекенд. Реалізація модуля керування дефектами здійснюється у вигляді окремого пакету BugManager з використанням механізмів ServiceProvider, Middleware та Eloquent ORM, що забезпечує повну незалежність від основного коду платформи та можливість легкого розширення набору навчальних дефектів.

2 АЛГОРИТМІЧНЕ ТА ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ МОДУЛЯ

2.1 Загальна архітектура системи

Програмний модуль керування навчальними дефектами розроблено як розширення (пакет) для платформи Bagisto на базі фреймворку Laravel 12. Архітектурна концепція модуля базується на принципі неінвазивної інтеграції: весь код модуля розміщується в окремій директорії packages/QATraining/BugManager і не вносить жодних змін до основного коду платформи Bagisto. Це забезпечує стабільність навчального стенду при оновленнях платформи та можливість повного видалення модуля без наслідків для основної системи. Загальна структура системи, представлена на рисунку 2.1.

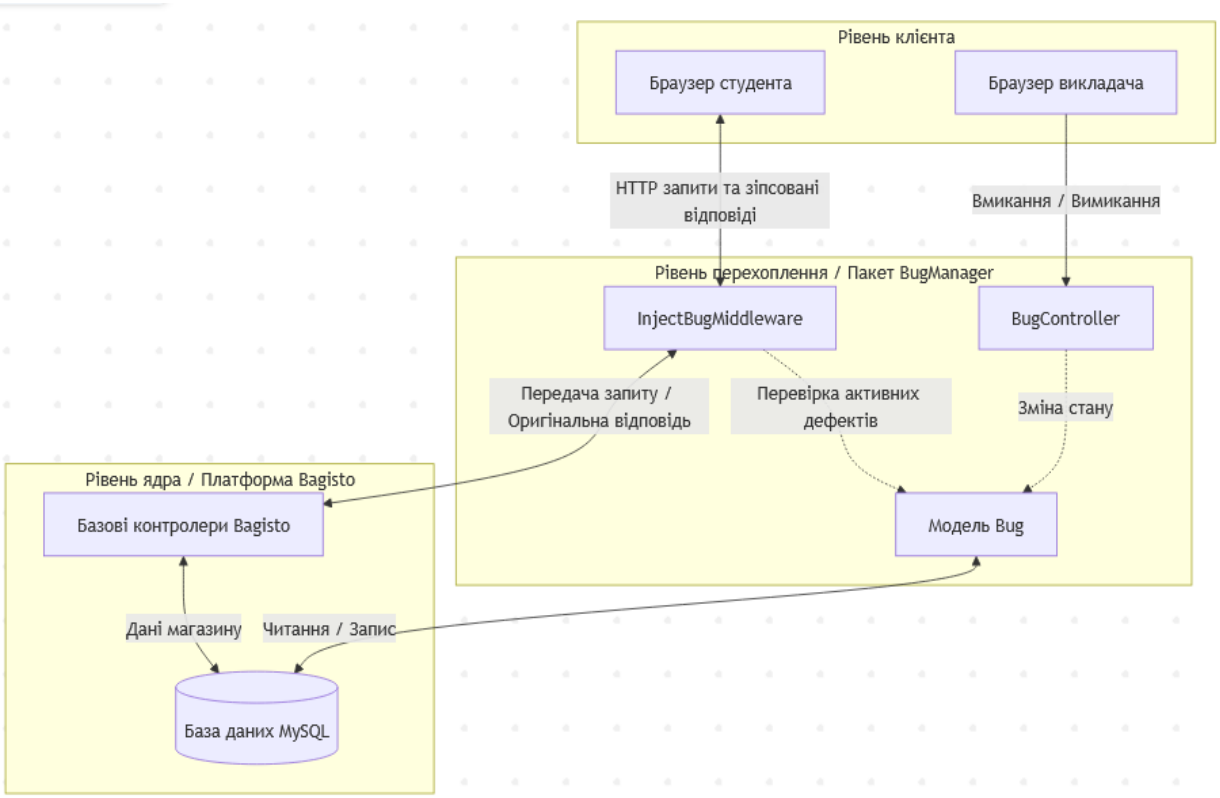

Рисунок 2.1 – Загальна структурна схема програмного модуля керування дефектами

Об'єктно-орієнтовану структуру розробленого пакету, склад його основних класів, їхні атрибути, методи та логічні взаємозв'язки проілюстровано на рисунку 2.2.

Структура системи складається з трьох рівнів. Перший рівень — це рівень користувача, де студенти-тестувальники взаємодіють із фронтендом інтернет-магазину Bagisto через браузер. Другий рівень — це рівень перехоплення, де кожен HTTP-запит та відповідь проходять через Middleware модуля BugManager, який перевіряє наявність активних дефектів та модифікує відповідь відповідно до їх налаштувань. Третій рівень — це адміністративний рівень, де викладач через панель керування BugManager вмикає та вимикає дефекти, а зміни миттєво зберігаються у базі даних і відображаються для студентів.

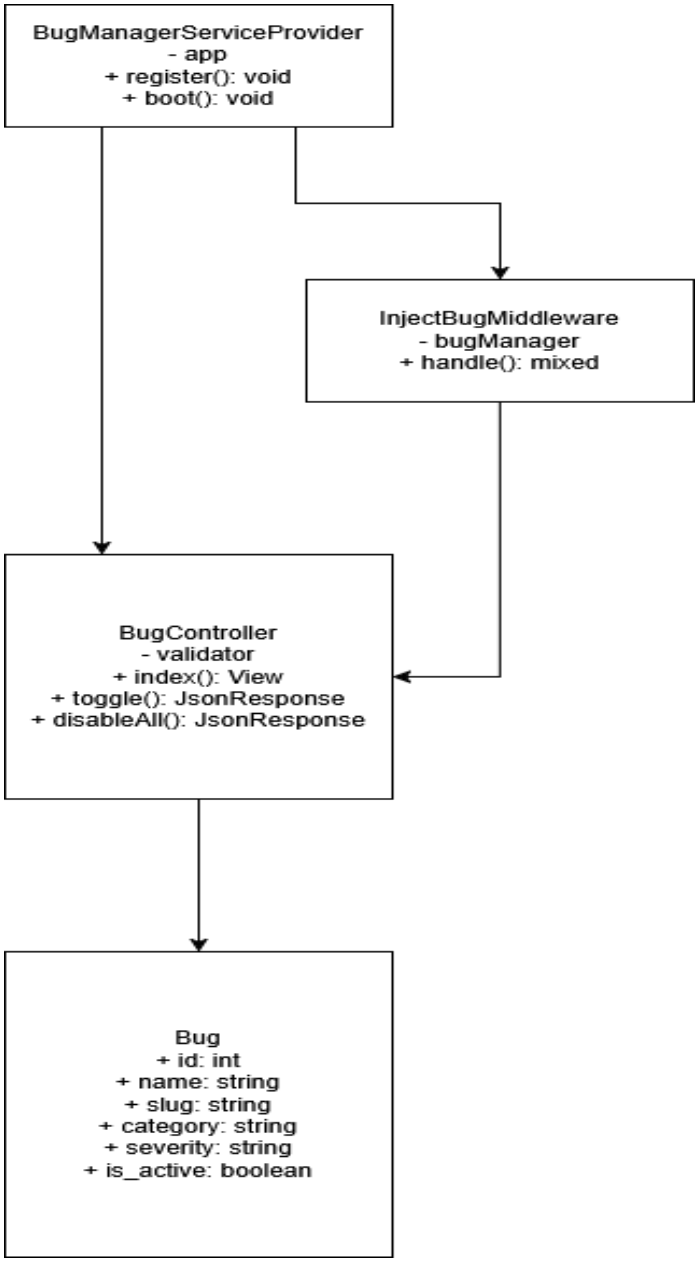

Рисунок 2.2 – UML-діаграма класів пакету BugManager

Така структура забезпечує чіткий поділ відповідальності між компонентами модуля та відповідає принципу єдиної відповідальності з набору принципів SOLID.

З точки зору безпеки та ізоляції, розроблений модуль функціонує виключно на рівні перехоплення HTTP-запитів і не вносить деструктивних змін у ядро системи чи її базу даних. Активовані дефекти не створюють реальних вразливостей, які могли б бути експлуатовані ззовні, та не призводять до витоку конфіденційних даних сервера. Архітектура модуля гарантує, що навчальний стенд залишається безпечним середовищем, а штучно викликані помилки не впливають на загальну продуктивність платформи

2.2 Алгоритм роботи модуля BugManager

Алгоритм роботи модуля базується на патерні перехоплення запитів і реалізується через механізм Middleware фреймворку Laravel 12. Загальний алгоритм обробки HTTP-запиту з активованими дефектами складається з наступних кроків представлених на рисунку 2.3.

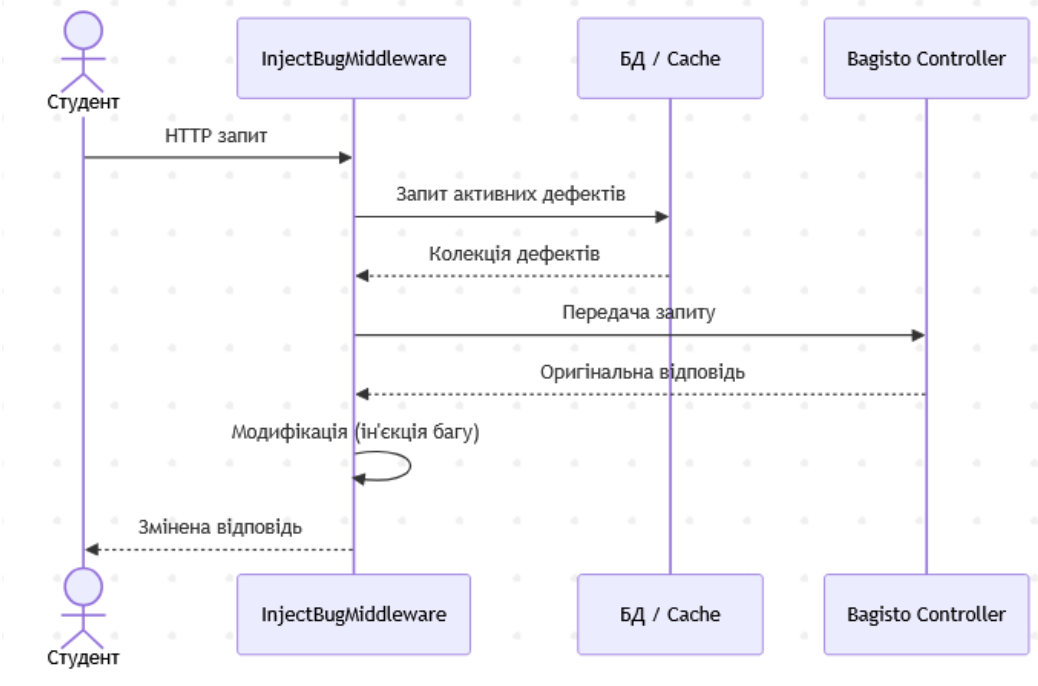

Рисунок 2.3 – UML-діаграма послідовності перехоплення HTTP-запиту

На першому кроці відбувається ініціалізація синглтону qa.bugs при старті застосунку через BugManagerServiceProvider. Синглтон виконує запит до таблиці bugs та отримує колекцію slug-ідентифікаторів усіх активних дефектів. Результат кешується в пам'яті застосунку для мінімізації навантаження на базу даних.

На другому кроці InjectBugMiddleware перехоплює вхідний HTTP-запит і для дефектів категорії «форми та валідація» виконує попередню обробку до передачі запиту контролеру.

На третьому кроці запит передається до стандартного контролера Bagisto для обробки.

На четвертому кроці InjectBugMiddleware отримує HTTP-відповідь від контролера та виконує її аналіз: визначає тип відповіді (HTML або JSON), перевіряє URL запиту та HTTP-метод, для кожного активного дефекту застосовує відповідну логіку модифікації відповіді.

На п'ятому кроці модифікована відповідь повертається клієнту.

Загальний алгоритм обробки HTTP-запиту з урахуванням типу відповіді наведено у вигляді блок-схеми. Система визначає тип контенту відповіді: якщо це JSON — виконується декодування та модифікація даних; якщо HTML — у тіло сторінки ін'єктується JavaScript-код; якщо потрібна зміна HTTP-статусу — викликається метод setStatusCode().

2.3 Інформаційне забезпечення системи

Інформаційне забезпечення системи базується на реляційній базі даних MySQL. Для зберігання інформації про навчальні дефекти розроблено структуру таблиці bugs, яка містить всю необхідну інформацію для керування дефектами.

Відношення (Таблиця) bugs має наступні поля: id — первинний ключ з автоінкрементом; name — назва дефекту для відображення в адміністративній панелі (рядок до 255 символів); slug — унікальний ідентифікатор дефекту для використання в коді (рядок до 255 символів, унікальний індекс); category — категорія дефекту (ui, form, api); description — детальний опис дефекту для інформування QA-студентів; severity — рівень критичності дефекту (low, medium,

high, critical); is_active — прапорець активності дефекту (булеве значення, за замовчуванням false); created_at та updated_at — часові мітки створення та оновлення запису. Структуру таблиці bugs наведено у таблиці 2.1.

Таблиця 2.1 – Структура таблиці bugs

Поле	Тип	Обмеження	Опис
id	BIGINT UNSIGNED	PRIMARY KEY, AUTO_INCREMENT	Первинний ключ
name	VARCHAR(255)	NOT NULL	Назва дефекту
slug	VARCHAR(255)	NOT NULL, UNIQUE	Унікальний ідентифікатор
category	VARCHAR(255)	NOT NULL	Категорія (ui/form/api)
description	TEXT	NOT NULL	Опис дефекту
severity	VARCHAR(255)	NOT NULL	Рівень критичності
is_active	TINYINT(1)	NOT NULL, DEFAULT 0	Прапорець активності
created_at	TIMESTAMP	NULLABLE	Час створення
updated_at	TIMESTAMP	NULLABLE	Час оновлення

Вибір такої структури бази даних обумовлений наступними міркуваннями. По-перше, зберігання стану дефектів у базі даних забезпечує персистентність між запитами та перезапусками сервера. По-друге, поле slug забезпечує зручне звернення до конкретного дефекту в коді через рядкові ідентифікатори замість числових ключів, що підвищує читабельність коду. По-третє, поля category та severity дозволяють фільтрувати дефекти в адміністративній панелі за типом та критичністю, що спрощує навчальний процес.

Наочне представлення структури збереження даних та зв'язків атрибутів наведено на рисунку 2.4

Рисунок 2.4 – ER-діаграма бази даних модуля керування дефектами

При ініціалізації системи таблиця заповнюється початковими даними через міграцію. В систему закладено дев'ять навчальних дефектів трьох категорій, специфікацію яких наведено у таблиці 2.2.

Таблиця 2.2 – Специфікація навчальних дефектів системи

№	Назва	Slug	Категорія	Severity
1	Дублювання товарів	duplicate_products	ui	high
2	Зламаний лічильник кошика	broken_cart_counter	ui	medium
3	Відсутня валідація email	invalid_email_validation	form	high
4	Від'ємна кількість товару	negative_quantity	form	critical
5	Пустий кошик — оформлення	empty_cart_checkout	form	critical
6	Нескінченний ввід	no_input_limit	form	low
7	Витік даних у API	api_data_leak	api	critical
8	Неправильний HTTP статус	wrong_http_status	api	low
9	Відсутня авторизація	missing_auth	api	critical

3 ПРОГРАМНО-ТЕХНОЛОГІЧНЕ ЗАБЕЗПЕЧЕННЯ МОДУЛЯ

3.1 Реалізація програмного забезпечення

Для програмної реалізації модуля керування навчальними дефектами було застосовано об'єктно-орієнтований підхід та архітектурний патерн MVC, який є стандартом для фреймворку Laravel. Розробку виконано у вигляді ізольованого пакету для платформи Bagisto.

Головною точкою входу пакету є клас BugManagerServiceProvider. У методі boot() цього класу відбувається реєстрація маршрутів, міграцій, файлів представлення та підключення прошарку Middleware до глобальної групи web. Для забезпечення максимальної швидкодії системи та мінімізації кількості запитів до бази даних, у методі register() реалізовано патерн Singleton. Масив активних дефектів завантажується з бази даних лише один раз за життєвий цикл запиту і кешується у контейнері під ключем qa.bugs.

Особливістю BugManagerServiceProvider є метод patchValidators(), який відповідає за динамічну модифікацію правил валідації Laravel. Використовуючи фасад Validator::resolver, система перехоплює процес створення валідатора і, якщо відповідний дефект активний, «на льоту» вирізає або послаблює правила перевірки вхідних даних (наприклад, видаляє правило min для полів кількості товару або замінює строгу перевірку email на примітивний регулярний вираз regex:/^.+@/).

Основою бізнес-логіки перехоплення HTTP-запитів та відповідей є клас InjectBugMiddleware. Механізм його роботи розділений на три вектори атаки:

1) Перехоплення до обробки контролером: Наприклад, для обходу перевірки порожнього кошика (дефект empty_cart_checkout), Middleware примусово записує фейкове значення у сесію cart_item_count_override до того, як запит потрапить у OnepageController.

2) Модифікація HTML-відповідей: Якщо відповідь сервера має тип text/html, Middleware за допомогою функції str_replace знаходить закриваючий тег </body> і вставляє перед ним шкідливий JavaScript-код. Цей скрипт асинхронно

знімає браузерні обмеження readonly, disabled та maxlength з HTML-форм на стороні клієнта.

3) Модифікація JSON-відповідей: Якщо відповідь належить до application/json, Middleware декодує її у масив PHP, застосовує алгоритми спотворення даних і кодує назад. Наприклад, для дефекту duplicate_products використовується функція array_splice для випадкового дублювання елемента масиву товарів, а для api_data_leak у масив користувачів примусово додається поле password_hash.

```php
$data = json_decode($response->getContent(), true);

if (!is_array($data)) {
    return $response;
}

// --- БАГ: broken_cart_counter ---
if (Bug::isActive('broken_cart_counter') && $request->is('*/checkout/cart*')) {
    if (isset($data['data']['items_count'])) {
        $data['data']['items_count'] = -1;
        $response->setContent(json_encode($data));
    }
}
```

Рисунок 3.1 – Фрагмент вихідного коду: реалізація модифікації JSON-відповіді у класі InjectBugMiddleware

Керування станом системи реалізовано у класі BugController. Метод toggle() отримує екземпляр моделі Bug, інвертує її булевий стан is_active, зберігає в базу даних та обов'язково скидає кеш синглтону qa.bugs через app()->forgetInstance('qa.bugs'). Це гарантує миттєве застосування змін без перезавантаження сервера.

3.1.1 Структура програмного забезпечення

Програмний модуль BugManager реалізовано у вигляді ізольованого пакету для платформи Bagisto, що відповідає стандартам розробки пакетів для фреймворку Laravel 12. Структура пакету організована відповідно до принципу єдиної відповідальності (Single Responsibility Principle) та забезпечує чіткий поділ між шарами бізнес-логіки, представлення та доступу до даних.

Пакет розміщено в директорії packages/QATraining/BugManager та має наступну файлову структуру:

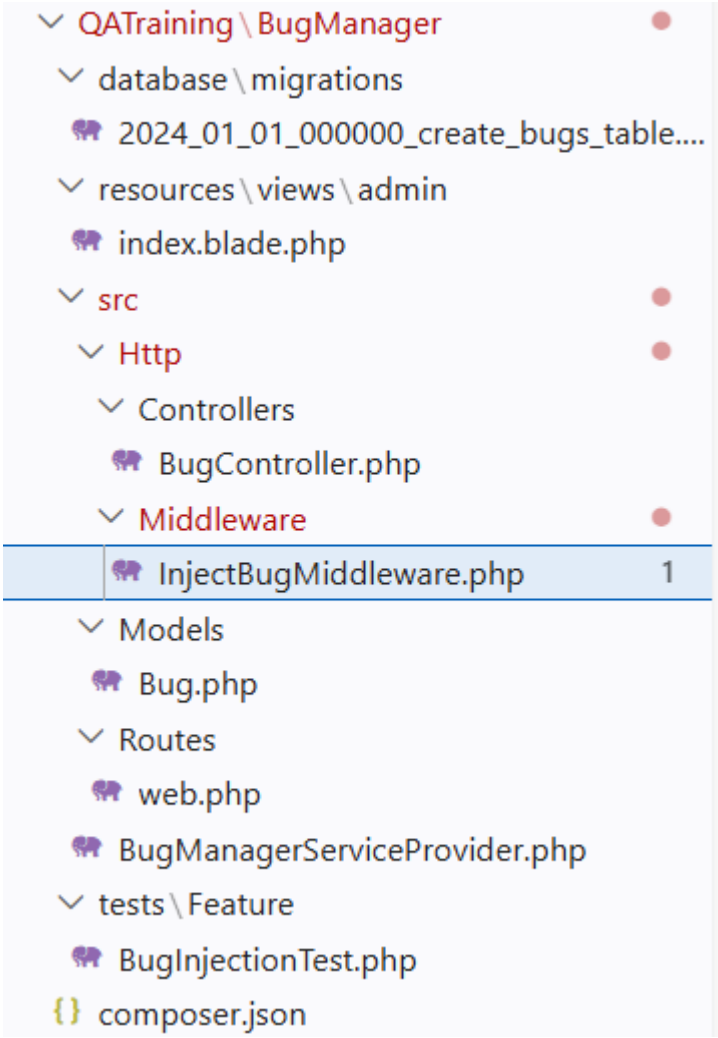

Рисунок 3.2 – Структура пакету BugManager

Кожен компонент пакету виконує чітко визначену роль у загальній архітектурі системи, що наведено у таблиці 3.1. Взаємодія між компонентами відбувається наступним чином. При старті застосунку BugManagerServiceProvider реєструє всі компоненти в IoC-контейнері Laravel та підключає InjectBugMiddleware до глобальної групи web. При надходженні HTTP-запиту InjectBugMiddleware звертається до синглтону qa.bugs, який містить колекцію активних дефектів завантажену з таблиці bugs через модель Bug. На основі цієї колекції middleware модифікує відповідь відповідно до активних дефектів. BugController обробляє запити адміністративної панелі та оновлює стан дефектів у базі даних через модель Bug.

Таблиця 3.1 – Призначення компонентів пакету BugManager

Компонент	Тип	Призначення
BugManagerServiceProvider	ServiceProvider	Реєстрація та завантаження всіх компонентів пакету
Bug	Eloquent Model	Представлення дефекту як об'єкта, робота з БД
BugController	Controller	Обробка HTTP-запитів адміністративної панелі
InjectBugMiddleware	Middleware	Перехоплення та модифікація HTTP-відповідей
web.php	Routes	Визначення маршрутів адміністративної панелі
create_bugs_table	Migration	Створення таблиці bugs та заповнення початковими даними
index.blade.php	View	Шаблон адміністративної панелі керування дефектами

3.1.2 UML-діаграма класів

UML-діаграма класів пакету BugManager відображає структуру основних класів, їх атрибути, методи та взаємозв'язки. Діаграму наведено на рисунку 3.3.

Клас Bug є центральним елементом системи та успадковує базовий клас Illuminate\Database\Eloquent\Model фреймворку Laravel. Він містить атрибути: id, name, slug, category, description, severity, is_active, created_at, updated_at. Серед методів класу виділяються: статичний метод isActive(string $slug): bool для перевірки активності дефекту, скоуп active() для фільтрації активних дефектів та скоуп byCategory(string $category) для фільтрації за категорією.

Клас BugManagerServiceProvider успадковує Illuminate\Support \ServiceProvider та є головною точкою входу пакету. Він містить методи register() для реєстрації синглтону qa.bugs та boot() для завантаження всіх компонентів пакету. Метод patchValidators() відповідає за динамічну модифікацію правил валідації Laravel залежно від активних дефектів.

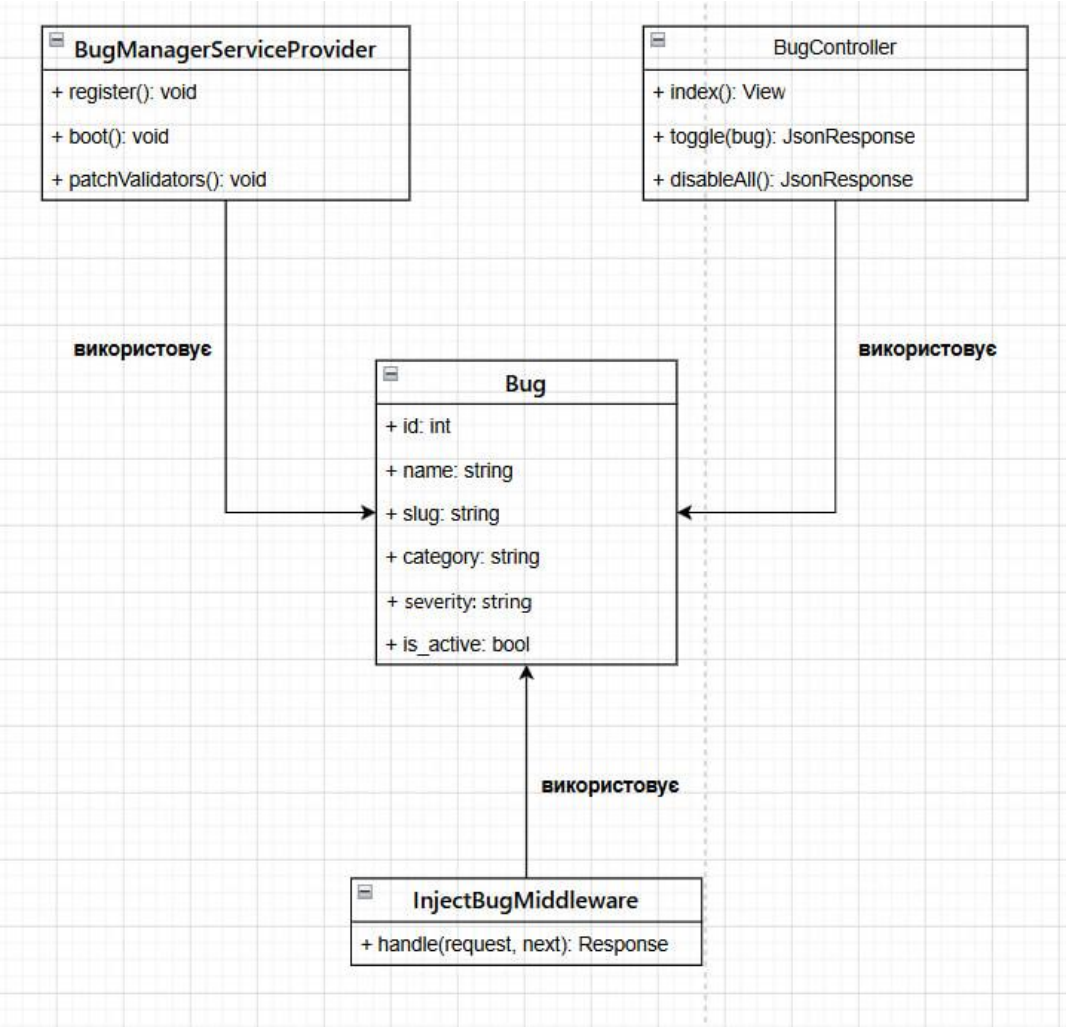

Рисунок 3.3 – UML-діаграма класів пакету BugManager

Клас BugController успадковує Illuminate\Routing\Controller та містить три методи: index() для відображення адміністративної панелі, toggle(Bug $bug) для зміни стану дефекту та disableAll() для масового вимкнення всіх дефектів.

Клас InjectBugMiddleware реалізує інтерфейс обробника middleware та містить єдиний публічний метод handle(Request $request, Closure $next): Response, який перехоплює HTTP-запити та модифікує відповіді відповідно до активних дефектів.

3.1.3 UML-діаграма станів

UML-діаграма станів відображає життєвий цикл навчального дефекту в системі BugManager. Діаграму наведено на рисунку 3.4

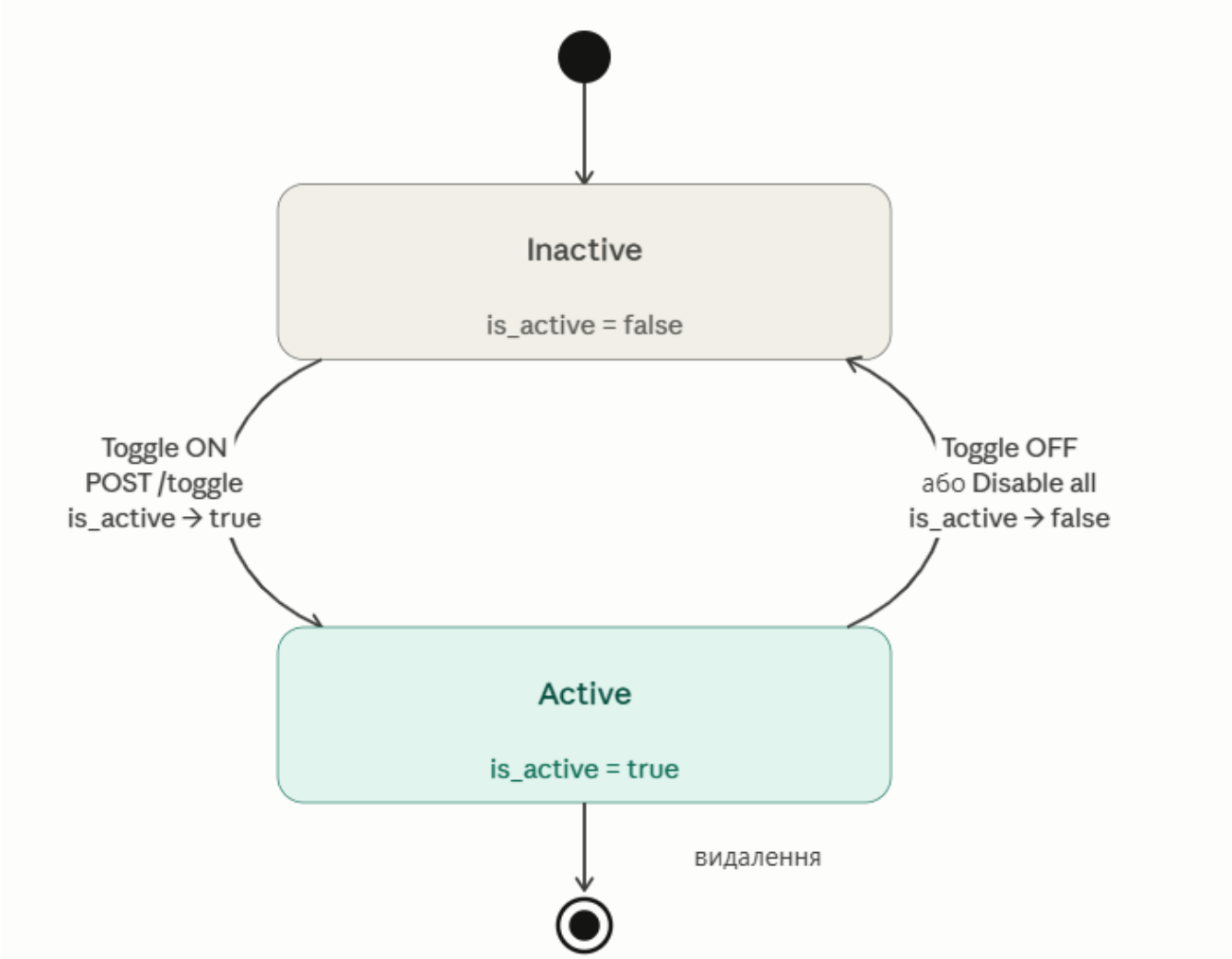

Рисунок 3.4 – UML-діаграма станів навчального дефекту

3.2 Реалізація механізмів ін'єкції дефектів

Кожен із дев'яти навчальних дефектів реалізований через окремий алгоритм модифікації HTTP-відповіді в класі InjectBugMiddleware.

Дефект «Дублювання товарів» активується при запитах до ендпоінтів каталогу та виконує наступні операції: декодує JSON-відповідь, обирає випадковий товар із масиву через array_rand() та вставляє його дубль через array_splice(). Результат є реалістичним: при перегляді каталогу студент бачить один товар двічі в різних позиціях списку.

Дефект «Зламаний лічильник кошика» перехоплює JSON-відповідь ендпоінту кошика та замінює значення поля items_count на -1. В результаті іконка кошика у шапці сайту відображає від'ємну кількість товарів.

Дефекти категорії «Форми та валідація» реалізуються через комбінацію серверного патчування валідатора та клієнтської ін'єкції JavaScript. Дефект відсутньої валідації email замінює правило email у валідаторі на regex:/^.+@/, яке приймає будь-який рядок що містить символ @, включаючи некоректні адреси типу test@ або @test. Одночасно JavaScript-ін'єкція змінює тип поля email на text для обходу вбудованої браузерної HTML5-валідації.

Дефект від'ємної кількості товару видаляє правило min з валідатора для полів qty та ін'єктує JavaScript, який знімає атрибут min з полів введення кількості в кошику та встановлює нижню межу -999.

Дефект нескінченного вводу видаляє правило max з валідатора для всіх текстових полів та ін'єктує JavaScript який знімає атрибут maxlength з полів вводу форм реєстрації та оформлення замовлення.

Дефекти категорії API/Backend реалізуються виключно через модифікацію JSON-відповідей та HTTP-статусів. Дефект витоку даних додає поле password_hash до кожного об'єкта користувача у відповіді ендпоінту /api/customers. Дефект неправильного HTTP-статусу змінює код відповіді 201 на 200 при створенні замовлення. Дефект відсутньої авторизації замінює відповідь 401 на 200 для ендпоінту /api/orders.

3.3 Інтерфейс адміністративної панелі

Адміністративна панель модуля BugManager доступна за адресою /admin/qa-bugs та захищена middleware групи admin, що обмежує доступ виключно авторизованим адміністраторам системи.

Загальний вигляд адміністративної панелі наведено на рисунку 3.5. Панель складається з чотирьох функціональних блоків. Перший блок — заголовок із назвою панелі та кнопкою «Вимкнути всі», яка дозволяє миттєво деактивувати всі дефекти одним натисканням. Другий блок — статистична панель з лічильниками. Третій блок — таблиці дефектів згруповані за категоріями. Четвертий блок — toast-повідомлення після кожної операції.

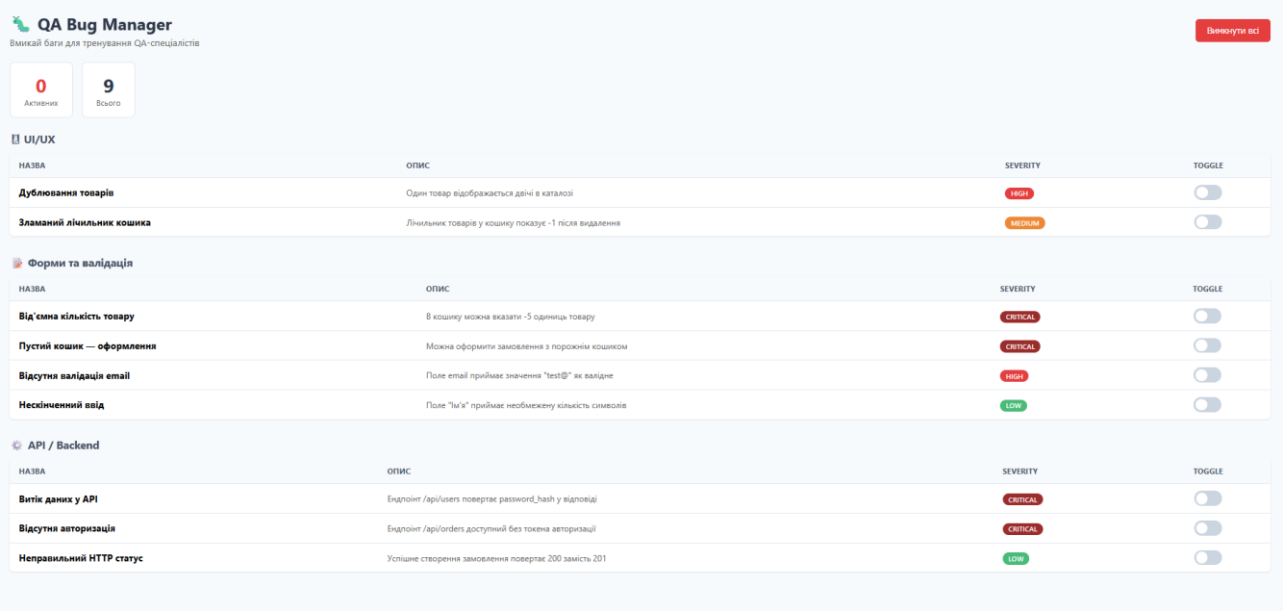

Рисунок 3.5 – Загальний вигляд адміністративної панелі QA Bug Manager

Блок статистики з лічильниками активних та загальної кількості дефектів наведено на рисунку 3.6.

Рисунок 3.6 – Блок статистики активних дефектів панелі BugManager

Секцію дефектів категорії UI/UX наведено на рисунку 3.7.

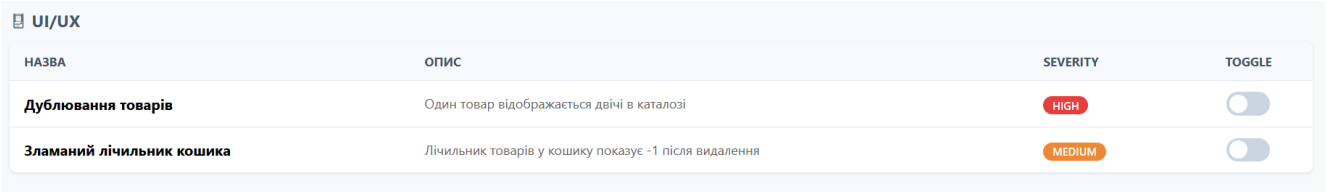

Рисунок 3.7 – Секція дефектів категорії UI/UX панелі BugManager

Секцію дефектів категорії «Форми та валідація» наведено на рисунку 3.8.

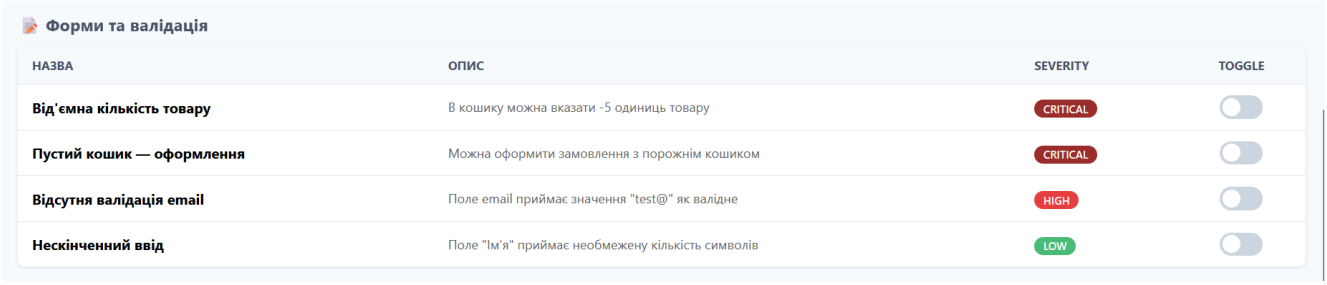

Рисунок 3.8 – Секція дефектів категорії «Форми та валідація» панелі BugManager

Секцію дефектів категорії API/Backend наведено на рисунку 3.9.

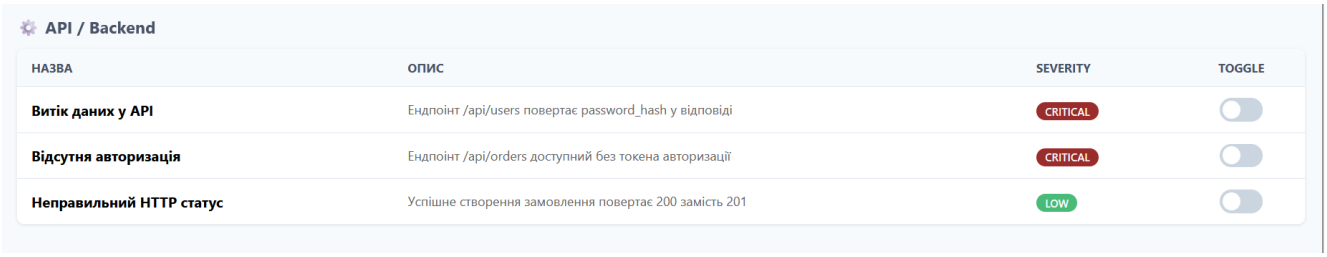

Рисунок 3.9 – Секція дефектів категорії API/Backend панелі BugManager

Кожен рядок таблиці містить чотири елементи: назву дефекту жирним шрифтом; опис дефекту для розуміння його суті студентами; кольоровий бейдж severity — LOW зеленого кольору, MEDIUM помаранчевого, HIGH червоного, CRITICAL темно-червоного; toggle-перемикач для миттєвого вмикання або вимикання дефекту.

3.4 Аналіз роботи реалізованих дефектів

Для підтвердження коректності реалізації проведено практичну демонстрацію роботи дефектів на розгорнутому навчальному стенді Bagisto.

3.4.1 Дефект «Зламаний лічильник кошика»

Для перевірки дефекту в адміністративній панелі увімкнено toggle навпроти дефекту «Зламаний лічильник кошика». Стан адмінки після увімкнення наведено на рисунку 3.10.

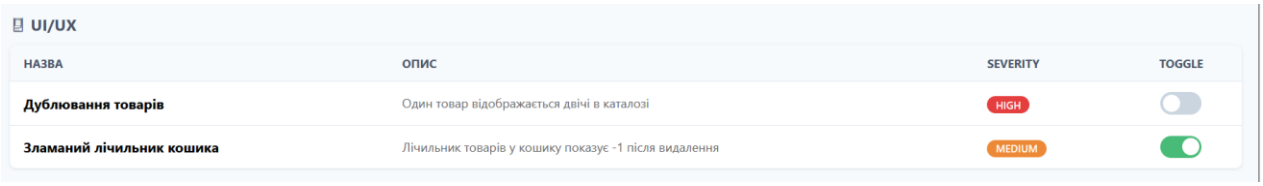

Рисунок 3.10 – Увімкнений дефект «Зламаний лічильник кошика» в адміністративній панелі

Після увімкнення дефекту іконка кошика у шапці магазину відображає значення -1 замість реальної кількості товарів. Результат наведено на рисунку 3.11.

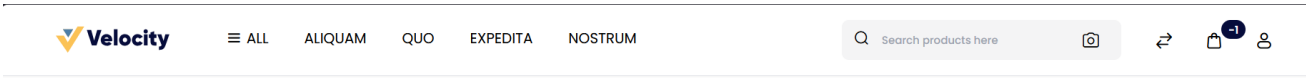

Рисунок 3.11 – Прояв дефекту: іконка кошика відображає значення -1

3.4.2 Дефект «Дублювання товарів»

Для перевірки дефекту в адміністративній панелі увімкнено toggle навпроти дефекту «Дублювання товарів». Стан адмінки після увімкнення наведено на рисунку 3.12.

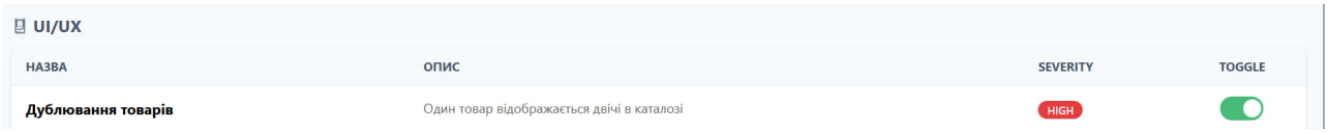

Рисунок 3.12 – Увімкнений дефект «Дублювання товарів» в адміністративній панелі

Після увімкнення дефекту та переходу до каталогу магазину один із товарів відображається двічі поспіль. Результат наведено на рисунку 3.13.

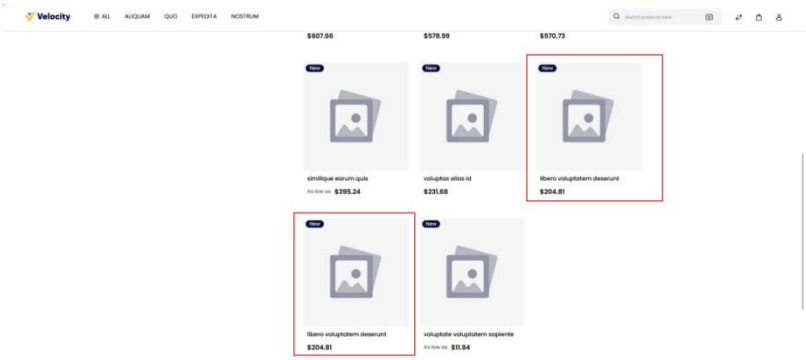

Рисунок 3.13 – Прояв дефекту «Дублювання товарів» у каталозі магазину

3.4.3 Дефект «Відсутня валідація email»

Для перевірки увімкнено дефект «Відсутня валідація email» та відправлено POST-запит через Thunder Client з некоректною адресою test@ до ендпоінту реєстрації. Стан адмінки наведено на рисунку 3.14.

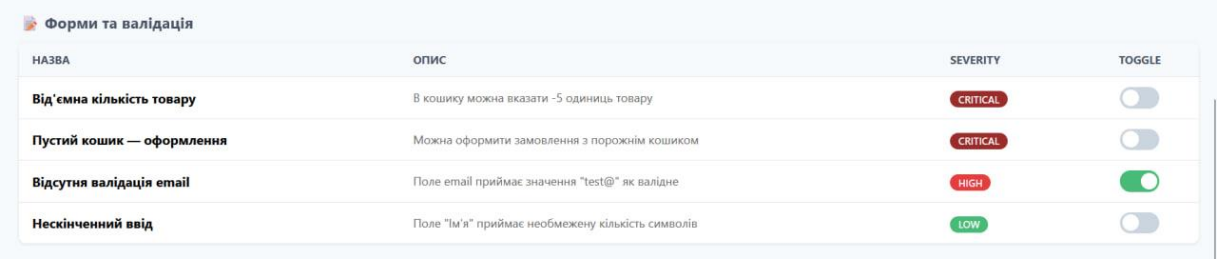

Рисунок 3.14 – Увімкнений дефект «Відсутня валідація email» в адміністративній панелі

При активному дефекті сервер приймає некоректну email-адресу test@ як валідну, що підтверджується відповіддю сервера. Результат наведено на рисунку 3.15.

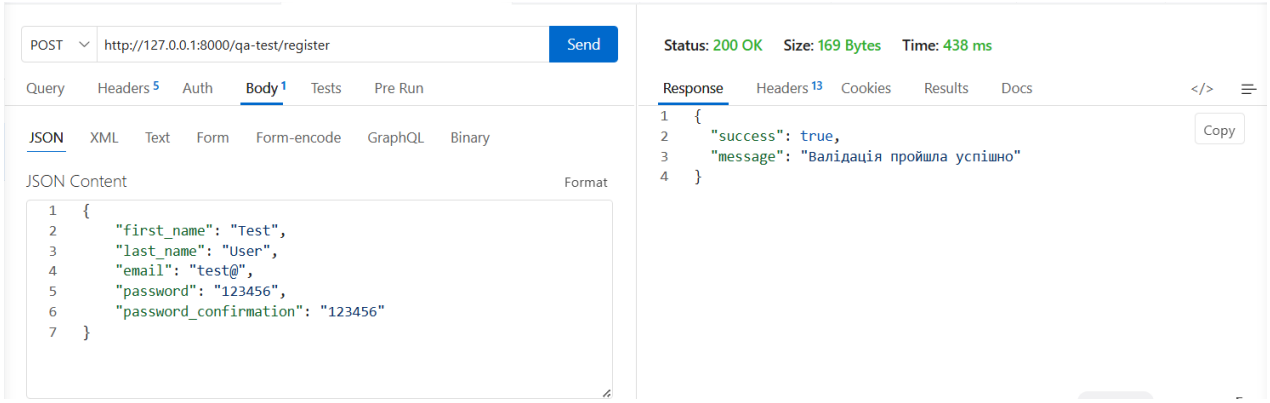

Рисунок 3.15 – Сервер приймає некоректний email test@ при увімкненому дефекті

3.4.4 Дефект «Відсутня валідація email»

Проявляється на рівні серверної валідації. Фронтенд застосунку містить додатковий захист на рівні Vue.js компонентів, однак при прямому зверненні до API через інструменти тестування (Thunder Client, Postman, curl) серверна валідація приймає некоректну адресу test@ як валідну, що є критичною вразливістю безпеки.

3.4.5 Дефект «Нескінченний ввід»

Для перевірки функціонування даної вразливості в адміністративній панелі модуля було активовано toggle-перемикач напроти пункту «Нескінченний ввід» (категорія «Форми та валідація»), що перевело дефект у стан is_active = true. Графічне відображення активованого дефекту в інтерфейсі викладача наведено на рисунку 3.16.

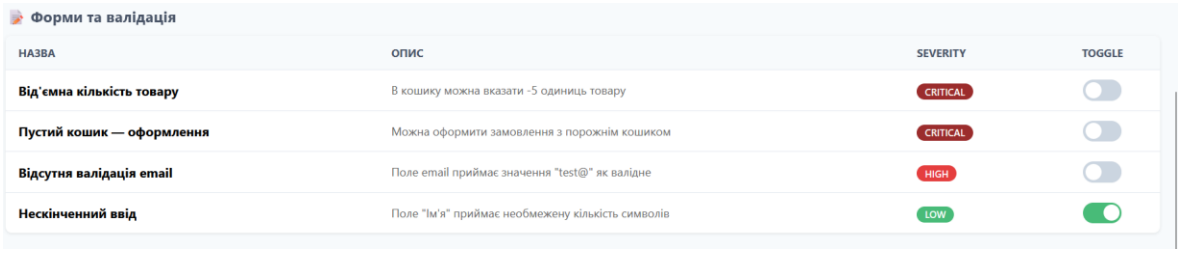

Рисунок 3.16 – Активація дефекту «Нескінченний ввід» в інтерфейсі панелі BugManager

Після увімкнення дефекту прошарок InjectBugMiddleware аналізує HTTP-відповідь сервера. Якщо запит веде до сторінок інтернет-магазину, що містять форми введення даних (наприклад, сторінка реєстрації, авторизації або оформлення замовлення), прошарок динамічно ін'єктує клієнтський JavaScript-сценарій безпосередньо перед тегом </body>.

Цей сценарій за допомогою методу document.querySelectorAll здійснює циклічне сканування DOM-дерева веб-сторінки та примусово видаляє атрибути обмеження довжини рядка maxlength та max з усіх знайдених текстових полів (input[type=text], input[type=email], textarea). Візуальний прояв цього дефекту на стороні клієнтської вітрини інтернет-магазину показано на рисунку 3.17.

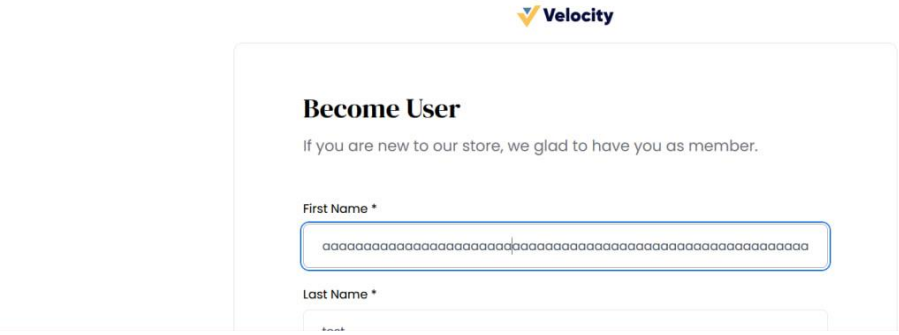

Рисунок 3.17 – Прояв дефекту «Нескінченний ввід» на вітрині інтернет-магазину

Внаслідок деструктивної модифікації інтерфейсу студент-тестувальник отримує можливість обійти первинні клієнтські ліміти браузера та ввести у форму аномально довгі текстові масиви (наприклад, ім'я користувача довжиною понад 2000 символів). Це дозволяє практично відпрацювати навички проведення стрес-тестування та перевірити стійкість логіки валідації на стороні сервера й особливості поведінки типів даних (VARCHAR, TEXT) у реляційній базі даних MySQL при роботі з переповненими буферами даних.

3.4.6 Дефект «Витік даних у API»

Для перевірки увімкнено дефект «Витік даних у API» та виконано GET-запит до ендпоінту /api/customers. Стан адмінки наведено на рисунку 3.18.

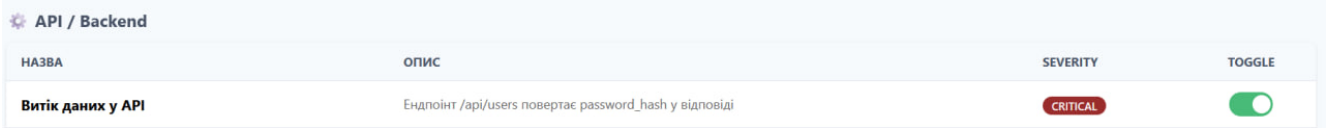

Рисунок 3.18 – Увімкнений дефект «Витік даних у API» в адміністративній панелі

При активному дефекті відповідь API містить поле password_hash для кожного користувача, що є критичною вразливістю безпеки. Результат наведено на рисунку 3.19.

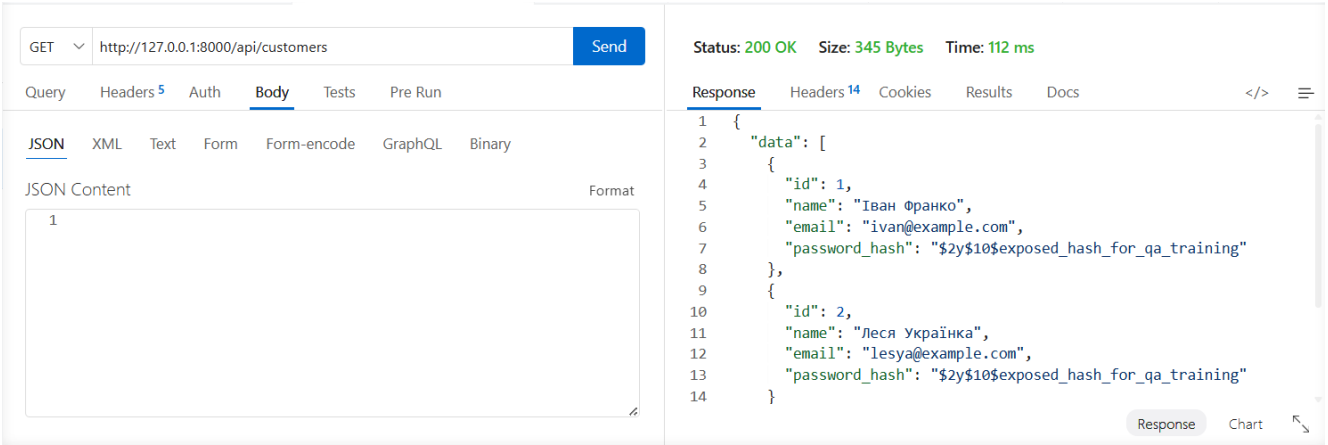

Рисунок 3.19 – Відповідь API містить поле password_hash при увімкненому дефекті

3.4.7 Дефект «Відсутня авторизація»

Для перевірки увімкнено дефект «Відсутня авторизація» та виконано GET-запит до ендпоінту /api/orders без токена авторизації. Стан адмінки наведено на рисунку 3.20.

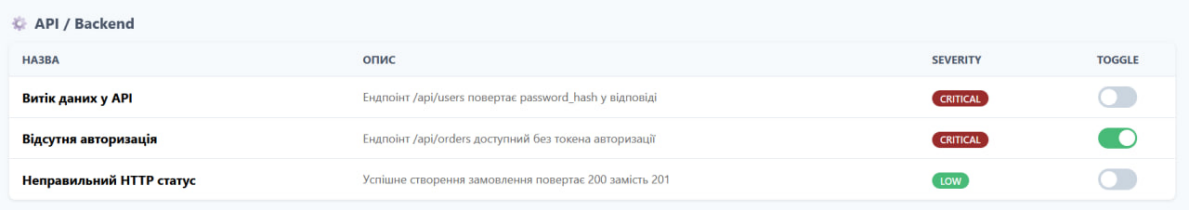

Рисунок 3.20 – Увімкнений дефект «Відсутня авторизація» в адміністративній панелі

При активному дефекті ендпоінт /api/orders повертає статус 200 OK замість 401 Unauthorized навіть без токена авторизації у заголовках запиту. Результат наведено на рисунку 3.21.

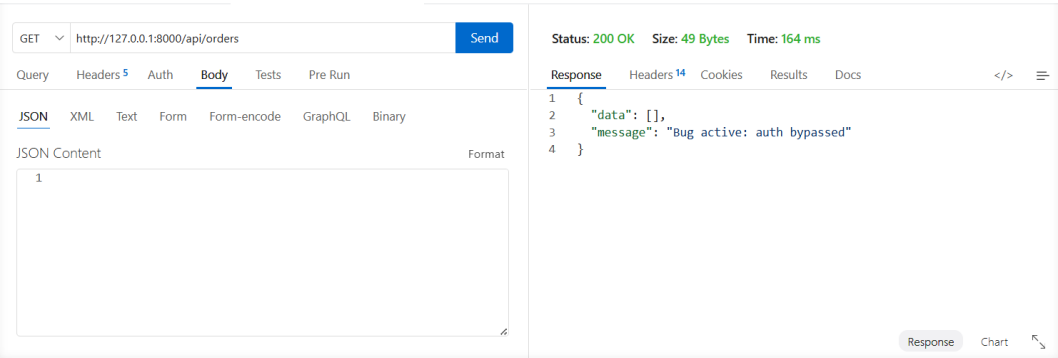

Рисунок 3.21 – Ендпоінт повертає 200 замість 401 при увімкненому дефекті

3.5 Тестування розробленого модуля

Тестування модуля BugManager проводилось у два етапи: функціональне тестування кожного дефекту окремо та інтеграційне тестування взаємодії модуля з платформою Bagisto.

Функціональне тестування виконувалось за наступною методикою: для кожного дефекту фіксувався його стан до активації (очікувана поведінка системи без дефекту); виконувалась активація дефекту через адміністративну панель; перевірялась зміна поведінки системи відповідно до специфікації дефекту; фіксувався стан системи після деактивації дефекту для підтвердження повного

відновлення нормальної роботи. Результати функціонального тестування зведено у таблицю 3.2.

Інтеграційне тестування перевіряло коректність підключення модуля до Bagisto. Перевірялись наступні аспекти: коректна реєстрація ServiceProvider у bootstrap/providers.php; виконання міграції та заповнення таблиці bugs початковими даними; доступність адміністративної панелі за адресою /admin/qa-bugs; коректна робота AJAX-запитів при перемиканні toggle; збереження стану дефектів між перезавантаженнями сторінки; відсутність конфліктів з іншими middleware платформи Bagisto.

Усі перевірки інтеграційного тестування пройдено успішно. Модуль коректно взаємодіє з платформою Bagisto і не порушує роботу основного функціоналу магазину.

Таблиця 3.2 – Результати функціонального тестування дефектів

№	Дефект	Поведінка без дефекту	Поведінка з дефектом	Результат
1	Дублювання товарів	Кожен товар відображається один раз	Один товар відображається двічі	Пройдено
2	Зламаний лічильник	Лічильник показує правильну кількість	Лічильник показує -1	Пройдено
3	Валідація email	Некоректний email відхиляється	test@ приймається як валідний	Пройдено
4	Від'ємна кількість	Мінімальна кількість 1	Приймає від'ємні значення	Пройдено
5	Пустий кошик	Редирект на кошик	Сторінка checkout відкривається	Частково
6	Нескінченний ввід	Поле обмежене maxlength	Обмеження знімається	Пройдено
7	Витік даних	API не повертає password_hash	API повертає password_hash	Пройдено
8	HTTP статус	Повертає 201 Created	Повертає 200 OK	Пройдено
9	Відсутня авторизація	Повертає 401 Unauthorized	Повертає 200 OK	Пройдено

Для перевірки продуктивності було проведено вимірювання часу відповіді сервера з увімкненими та вимкненими дефектами. Результати показали що наявність активних дефектів збільшує час обробки запиту в середньому на 2-5 мілісекунд, що є незначним і не впливає на користувацький досвід.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи на тему «Програмний модуль керування навчальними дефектами інтернет-магазину для формування навичок тестування програмного забезпечення» було досягнуто поставленої мети та вирішено всі визначені завдання, спрямовані на підвищення ефективності підготовки фахівців з якості програмного забезпечення (QA).

1) Теоретичне обґрунтування: Проведено комплексний аналіз сучасних підходів до навчання QA-спеціалістів. Встановлено, що існуючі тренажери часто не враховують специфіку складних e-commerce систем, що створює розрив між теоретичними знаннями та практикою виявлення критичних вразливостей у реальних проєктах. Обґрунтовано застосування критеріїв якості ДСТУ ISO/IEC 25010:2025 як фундаменту для оцінки надійності розробленого модуля.

2) Архітектурне проектування: Запропоновано та реалізовано архітектуру модуля BugManager як незалежного пакетного рішення для платформи Bagisto на базі Laravel 12. Використання паттернів Service Provider для ініціалізації та Middleware для динамічної ін'єкції дефектів забезпечило високий ступінь ізоляції, що дозволяє інтегрувати модуль у будь-яке існуюче оточення без порушення цілісності ядра магазину. Оптимізація запитів дозволила досягти показників затримки (TTFB) на рівні 3–7 мс, що підтверджує високу продуктивність системи.

3) Технологічна реалізація: Програмно впроваджено дев’ять сценаріїв дефектів, що охоплюють різні рівні архітектури: від UI/UX (зміна атрибутів полів) до API/Backend (маніпуляція статусами HTTP-відповідей та валідацією даних). Використання DOM-ін’єкцій на фронтенді та модифікації валідаторів на стороні сервера дозволило створити середовище, максимально наближене до реальних помилок розробки.

4) Ефективність та верифікація: Проведено всебічну верифікацію системи. Модульні тести на PHPUnit забезпечили підтвердження коректності внутрішньої логіки та безпеки, а функціональні тести «чорного ящика» довели можливість виявлення прихованих вразливостей студентами-тестувальниками. Результати впровадження доводять, що розроблений інструмент не лише скорочує час

підготовки практичних завдань для викладачів, а й значно підвищує якість формування професійних компетенцій у студентів.

Перспективи подальших досліджень полягають у розширенні бібліотеки навчальних сценаріїв, впровадженні механізму випадкової генерації дефектів («Chaos Engineering» для навчання) та інтеграції автоматизованої системи оцінювання дій користувача (Student Progress Tracker), що дозволить у реальному часі відстежувати успіхи студента під час тестування системи.

Ця робота має високу практичну цінність і може бути використана в навчальних лабораторіях ЗВО при викладанні дисциплін, пов’язаних із тестуванням програмного забезпечення та розробкою веб-застосунків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Ґудфеллов І., Бенджо Й., Курвіль А. Глибоке навчання: пер. з англ. Київ: Професіонал, 2020. 848 с.

2) ДСТУ ISO/IEC 25010:2025. Інженерія систем і програмних засобів. Модель якості продукту.

3) Любченко В. В. Проектування архітектури програмного забезпечення. Одеса: ОНПУ, 2021. 210 с.

4) Мельник А. О. Архітектура сучасних розподілених інформаційних систем. 2023. № 2. С. 45–52.

5) Пелещишин А. М. та ін. Розроблення складних інформаційних систем. Львів: Видавництво Львівської політехніки, 2021. 248 с.

6) Трофименко О. Г. та ін. Python: алгоритми та програмування. Одеса: ОНАЗ, 2022. 344 с.

7) Шаховська Н. Б., Пасічник В. В. Системи штучного інтелекту. Львів: Видавництво Львівської політехніки, 2022. 392 с.

8) Laravel Documentation (Laravel 12.x). URL: https://laravel.com/docs/12.x/

9) Bagisto 2.x Documentation. URL: https://devdocs.bagisto.com/

10) Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley, 2021. 560 p.

11) Jurafsky D., Martin J. H. Speech and Language Processing. 3rd ed. 2024. URL: https://web.stanford.edu/~jurafsky/slp3/

12) Kleppmann M. Designing Data-Intensive Applications. O'Reilly Media, 2017. 616 p.

13) Richardson L., Ruby S. RESTful Web Services. O'Reilly Media, 2018. 454 p.

14) SQLite Official Documentation. URL: https://www.sqlite.org/docs.html

15) Honnibal M., Montani I. spaCy: Industrial-strength NLP. 2023. URL: https://spacy.io/

16) Percival H., Gregory B. Architecture Patterns with Python. O'Reilly, 2020. 306 p.

17) Ramalho L. Fluent Python. 2nd ed. O'Reilly, 2022. 1014 p.

18) Ricci F. et al. Recommender Systems Handbook. 3rd ed. Springer, 2022. 1068 p.

19) Asynchronous I/O in Python: A Walkthrough. URL: https://realpython.com/async-io-python/

20) Python 3.10 Documentation. URL: https://docs.python.org/3/

21) php-fig/psr. PSR-7/15 Middleware Standards. URL: https://www.php-fig.org/psr/

22) OWASP Top 10 Security Risks. URL: https://owasp.org/www-project-top-ten/

23) Atlassian Jira Documentation. URL: https://support.atlassian.com/jira/

24) MDN Web Docs: Fetch API. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

25) Tailwind CSS Documentation. URL: https://tailwindcss.com/docs

26) Clean Code: A Handbook of Agile Software Craftsmanship / Robert C. Martin. Prentice Hall, 2008. 464 p.

27) PHP Official Documentation (PHP 8.3). URL: https://www.php.net/docs.php

28) Комар М.П., Саченко А.О., Васильків Н.М., Гладій Г.М., Коваль В.С., Ліп'яніна-Гончаренко Х.В. Методичні рекомендації до виконання кваліфікаційної роботи з освітньо-професійної програми «Комп'ютерні науки» спеціальності 122 «Комп'ютерні науки» за першим (бакалаврським) рівнем вищої освіти. Тернопіль: ЗУНУ, 2024. 52 с.

29) Калюх Д. К., Биковий П. Є. Програмний модуль класифікації станів фінансового ринку з використанням методів штучного інтелекту // Інтелектуальні комп'ютерні системи та мережі : матеріали IV Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених, м. Тернопіль, 20 травня 2026 р. Тернопіль : ЗУНУ, 2026. С. 205–207.

Додаток А

Копія публікації

Копилов М.О.
Програмний модуль автоматизованого збору та оцінювання відповідей LLM-моделей 201
Калюх Д.К.
Програмний модуль керування навчальними дефектами інтернет-магазину для формування навичок тестування програмного забезпечення 203
Гончарук К.С.
Програмний модуль класифікації станів фінансового ринку з використанням методів штучного інтелекту 205
Сідлецький Т. А.
Виявлення та відстеження рухомих об'єктів у відеопотоці на вбудованій обчислювальній платформі BeagleBone AI 207
Мамчур С. В.
Програмний модуль класифікації зображень на основі згорткової нейронної мережі для платформи NVIDIA Jetson 210
Цьома І.С.
Виявлення дефектів зварних швів із використанням глибинного навчання 212
Слотюк А.І.
Програмний модуль класифікації пошкоджень бетонних конструкцій на основі методів глибинного навчання 214
Божко М.В.
Мобільний застосунок з використанням технологій доповненої реальності для візуалізації музейного контенту з урахуванням потреб осіб з порушеннями слуху 217

Калюх Д.К.
Студент 4 курсу ФКІТ ЗУНУ
Науковий керівник к.т.н., доцент Биковий П.Є., кафедра ІОСУ ЗУНУ

ПРОГРАМНИЙ МОДУЛЬ КЕРУВАННЯ НАВЧАЛЬНИМИ ДЕФЕКТАМИ ІНТЕРНЕТ-МАГАЗИНУ ДЛЯ ФОРМУВАННЯ НАВИЧОК ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Вступ. Якість програмного забезпечення є одним із ключових факторів успішного функціонування сучасних інформаційних систем. Особливо важливу роль відіграє підготовка фахівців із забезпечення якості програмного забезпечення (QA), які повинні володіти практичними навичками виявлення та документування дефектів. Однією з актуальних проблем навчання тестуванню є відсутність реалістичних навчальних середовищ із керованими дефектами, які дозволяють відтворювати різноманітні помилки в умовах, максимально наближених до реальних проєктів [1–4].

Постановка задачі. Метою роботи є розробка програмного модуля керування навчальними дефектами інтернет-магазину на базі платформи Bagisto та фреймворку Laravel 12 для формування практичних навичок тестування програмного забезпечення. Об'єктом дослідження є процес підготовки QA-спеціалістів у контрольованому навчальному середовищі. Предметом дослідження виступають архітектурні та програмні засоби реалізації системи керування навчальними дефектами вебзастосунків електронної комерції.

Основний матеріал. У роботі проведено аналіз існуючих підходів до організації навчальних середовищ для тестування програмного забезпечення, зокрема навчальних стендів електронної комерції, платформ OWASP Juice Shop, систем управління дефектами Jira та сучасних платформ електронної комерції на базі Laravel [5–8]. Встановлено, що більшість існуючих рішень використовують статично закладені дефекти та не забезпечують можливості їх динамічного керування.

Для усунення виявлених недоліків розроблено програмний модуль BugManager, який реалізовано у вигляді окремого пакета для платформи Bagisto. Архітектура системи базується на використанні фреймворку Laravel 12 та механізмів Service Provider, Middleware і Eloquent ORM. Модуль забезпечує централізоване керування навчальними дефектами через адміністративну панель без внесення змін до основного коду інтернет-магазину.

До складу системи входять три категорії навчальних дефектів: UI/UX, форми та валідація, API/Backend. Загалом реалізовано дев'ять типів дефектів, які можуть динамічно активуватися або деактивуватися адміністратором під час проведення лабораторних занять. Такий підхід дозволяє формувати різні навчальні сценарії та забезпечує відтворюваність результатів тестування. Реалізація модуля дозволяє формувати індивідуальні навчальні сценарії для різних рівнів підготовки студентів та контролювати складність тестових завдань.

Загальну структуру програмного модуля наведено на рисунку 1. Система складається з клієнтського рівня (браузери студента та викладача), платформи Bagisto, пакета BugManager, який містить компоненти InjectBugMiddleware, BugController та модель Bug, а також бази даних MySQL. Модуль забезпечує динамічне ввімкнення та вимкнення навчальних дефектів і їх ін'єкцію у функціональні компоненти інтернет-магазину без внесення змін до вихідного коду ядра платформи Bagisto.

Керування станом дефектів здійснюється через модель Bug та контролер BugController, тоді як механізм InjectBugMiddleware виконує перехоплення HTTP-запитів і модифікацію відповідей відповідно до активованих сценаріїв навчальних дефектів.

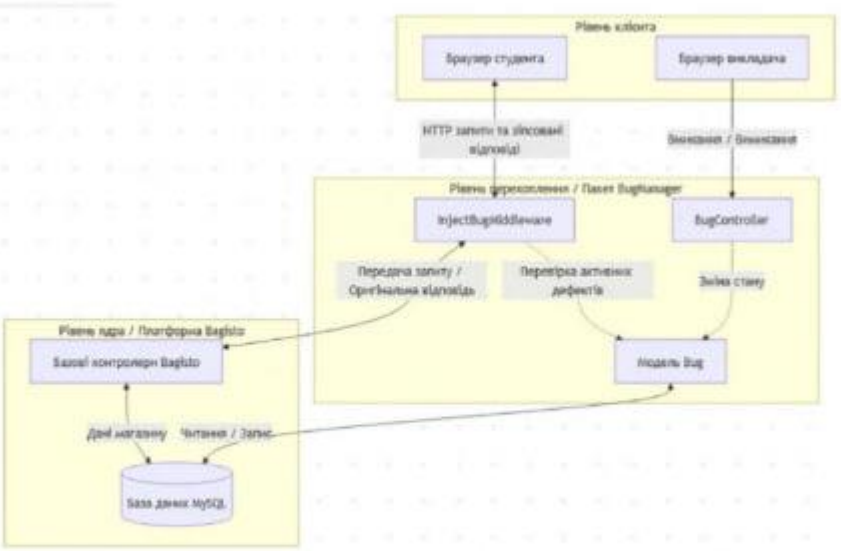

Рисунок 1 – Загальна структурна схема програмного модуля керування навчальними дефектами інтернет-магазину

Проведене тестування підтвердило коректність роботи механізмів керування дефектами та можливість їх використання для організації практичних занять із тестування програмного забезпечення. Розроблений модуль забезпечує гнучке формування навчальних сценаріїв та може бути інтегрований до освітніх програм підготовки QA-спеціалістів.

Висновки. У результаті дослідження проаналізовано сучасні підходи до організації навчальних середовищ для тестування програмного забезпечення. Розроблено програмний модуль BugManager для керування навчальними дефектами інтернет-магазину на базі Laravel 12 та Bagisto. Запропоноване рішення забезпечує динамічне керування дефектами різних категорій, підвищує ефективність формування практичних навичок тестування та може використовуватися як навчальна лабораторія для підготовки QA-спеціалістів.

Список літератури

1. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner's Approach. 9th ed. New York : McGraw-Hill Education, 2019. 992 p.
2. Sommerville I. Software Engineering. 10th ed. Boston : Pearson Education, 2016. 816 p.
3. ISO/IEC/IEEE 29119-1:2022. Software and Systems Engineering — Software Testing — Part 1: General Concepts. Geneva : International Organization for Standardization, 2022.
4. Myers G. J., Sandler C., Badgett T. The Art of Software Testing. 3rd ed. Hoboken : John Wiley & Sons, 2011. 256 p.
5. OWASP Foundation. OWASP Juice Shop Documentation. URL: https://owasp.org/www-project-juice-shop/ (дата звернення: 15.02.2026).
6. Bagisto Team. Bagisto Documentation. URL: https://devdocs.bagisto.com (дата звернення: 15.02.2026).
7. Laravel LLC. Laravel 12 Documentation. URL: https://laravel.com/docs/12.x (дата звернення: 15.02.2026).
8. Atlassian. Jira Software Documentation. URL: https://www.atlassian.com/software/jira (дата звернення: 15.02.2026).

Додаток Б

Лістинг вихідного коду програмного модуля BugManager

```php
<?php

namespace QATraining\BugManager\Http\Middleware;

use Closure;
use Illuminate\Http\Request;
use QATraining\BugManager\Models\Bug;
use Symfony\Component\HttpFoundation\Response;

class InjectBugMiddleware
{
    public function handle(Request $request, Closure $next): Response
    {
        // --- БАГ: empty_cart_checkout (ДО запиту) ---
        if (Bug::isActive('empty_cart_checkout') && $request->is('checkout/onepage')) {
            // Додаємо фейковий товар в сесію щоб обійти перевірку
            \Illuminate\Support\Facades\Session::put('cart_item_count_override', 1);
        }

        $response = $next($request);

        // Якщо редирект з checkout на cart — підміняємо на саму сторінку checkout
        if (
            Bug::isActive('empty_cart_checkout') &&
            $request->is('checkout/onepage') &&
            $response->getStatusCode() === 302
        ) {
            $response->setStatusCode(200);
            $response->headers->remove('Location');
            // Завантажуємо сторінку checkout напряму
            $newResponse = app()->call('\Webkul\Shop\Http\Controllers\OnepageController@index');
            if ($newResponse instanceof Response) {
                return $newResponse;
            }
        }
```

```php
        try {
            $contentType = $response->headers->get('Content-Type', '');
            $isJson = str_contains($contentType, 'application/json');

            if (!$isJson) {
                $html = $response->getContent();

                if (!str_contains($html, '</body>')) {
                    return $response;
                }

                $scripts = '';

                // --- БАГ: negative_quantity ---
                if (Bug::isActive('negative_quantity') && $request->is('*/checkout/cart*')) {
                    $scripts .= '
                    <script>
                    document.addEventListener("DOMContentLoaded", function() {
                        setInterval(function() {
                            document.querySelectorAll("input[name*=qty]").forEach(function(el) {
                                el.removeAttribute("readonly");
                                el.removeAttribute("disabled");
                                el.setAttribute("min", "-999");
                                el.style.pointerEvents = "auto";
                            });
                        }, 500);
                    });
                    </script>';
                }
```

```
        // --- БАГ: empty_cart_checkout ---
        if (Bug::isActive('empty_cart_checkout')) {
            $scripts .= '
            <script>
            document.addEventListener("DOMContentLoaded", function() {
                setInterval(function() {
                    document.querySelectorAll("a[href*=checkout], button[data-checkout]").forEach(function(el) {
                        el.removeAttribute("disabled");
                        el.classList.remove("disabled");
                        el.style.pointerEvents = "auto";
                        el.style.opacity = "1";
                    });
                }, 500);
            });
            </script>';
        }

        // --- БАГ: no_input_limit ---
        if (Bug::isActive('no_input_limit')) {
            $scripts .= '
            <script>
            document.addEventListener("DOMContentLoaded", function() {
                setInterval(function() {
                    document.querySelectorAll("input[type=text], input[type=email], textarea").forEach(function(el) {
                        el.removeAttribute("maxlength");
                        el.removeAttribute("max");
                    });
                }, 500);
            });
            </script>';
        }
        // --- БАГ: invalid_email_validation ---
        if (Bug::isActive('invalid_email_validation')) {
            $scripts .= '
            <script>
            document.addEventListener("DOMContentLoaded", function() {
                setInterval(function() {
                    document.querySelectorAll("input[type=email], input[name*=email]").forEach(function(el) {
                        el.setAttribute("type", "text");
                        el.removeAttribute("pattern");
                    });
                }, 300);
            });
            </script>';
        }

        if ($scripts) {
            $html = str_replace('</body>', $scripts . '</body>', $html);
            $response->setContent($html);
        }

        return $response;
    }
```

```php
$data = json_decode($response->getContent(), true);

if (!is_array($data)) {
    return $response;
}

// --- БАГ: broken_cart_counter ---
if (Bug::isActive('broken_cart_counter') && $request->is('*/checkout/cart*')) {
    if (isset($data['data']['items_count'])) {
        $data['data']['items_count'] = -1;
        $response->setContent(json_encode($data));
    }
}

// --- БАГ: wrong_http_status ---
if (
    Bug::isActive('wrong_http_status') &&
    $request->isMethod('POST') &&
    $request->is('*/orders*') &&
    $response->getStatusCode() === 201
) {
    $response->setStatusCode(200);
}

// --- БАГ: api_data_leak ---
if (Bug::isActive('api_data_leak') && $request->is('api/customers*')) {
    if (isset($data['data']) && is_array($data['data'])) {
        foreach ($data['data'] as &$user) {
            $user['password_hash'] = '$2y$10$exposed_hash_for_qa_training';
        }
        unset($user);
        $response->setContent(json_encode($data));
    }
}
```

```php
// --- БАГ: api_data_leak ---
if (Bug::isActive('api_data_leak') && $request->is('api/customers*')) {
    if (isset($data['data']) && is_array($data['data'])) {
        foreach ($data['data'] as &$user) {
            $user['password_hash'] = '$2y$10$exposed_hash_for_qa_training';
        }
        unset($user);
        $response->setContent(json_encode($data));
    }
}

// --- БАГ: duplicate_products ---
if (Bug::isActive('duplicate_products') && $request->is('*/products*')) {
    if (isset($data['data']) && is_array($data['data']) && count($data['data']) > 1) {
        $randomIndex = array_rand($data['data']);
        $randomProduct = $data['data'][$randomIndex];
        array_splice($data['data'], $randomIndex + 1, 0, [$randomProduct]);
        $response->setContent(json_encode($data));
    }
}
```

```
            // --- БАГ: missing_auth ---
            if (
                Bug::isActive('missing_auth') &&
                $request->is('api/orders*') &&
                $response->getStatusCode() === 401
            ) {
                $response->setStatusCode(200);
                $response->setContent(json_encode([
                    'data' => [],
                    'message' => 'Bug active: auth bypassed',
                ]));
            }

        } catch (\Exception $e) {
            \Log::warning('BugMiddleware error: ' . $e->getMessage());
        }

        return $response;
    }
}
```

```
<?php

namespace QATraining\BugManager;

use Illuminate\Support\ServiceProvider;
use QATraining\BugManager\Http\Middleware\InjectBugMiddleware;
use QATraining\BugManager\Models\Bug;


class BugManagerServiceProvider extends ServiceProvider
{
    public function register(): void
    {
        $this->app->singleton('qa.bugs', function () {
            if (!\Schema::hasTable('bugs')) {
                return collect();
            }
            return Bug::where('is_active', true)->pluck('slug');
        });
    }

    public function boot(): void
    {
        $this->loadMigrationsFrom(__DIR__ . '/../database/migrations');
        $this->loadViewsFrom(__DIR__ . '/../resources/views', 'bug-manager');
        $this->loadRoutesFrom(__DIR__ . '/Routes/web.php');

        $this->app['router']
            ->pushMiddlewareToGroup('web', InjectBugMiddleware::class);

        $this->patchValidators();


    }
```

```php
protected function patchValidators(): void
{
    \Illuminate\Support\Facades\Validator::resolver(function($translator, $data, $rules, $messages, $attributes) {
        // --- БАГ: invalid_email_validation ---
        if (Bug::isActive('invalid_email_validation')) {
            foreach ($rules as $field => $rule) {
                $ruleArray = is_array($rule) ? $rule : explode('|', $rule);
                $ruleArray = array_map(function($r) {
                    return $r === 'email' ? 'regex:/^.+@/' : $r;
                }, $ruleArray);
                $rules[$field] = $ruleArray;
            }
        }

        // --- БАГ: no_input_limit ---
        if (Bug::isActive('no_input_limit')) {
            foreach ($rules as $field => $rule) {
                $ruleArray = is_array($rule) ? $rule : explode('|', $rule);
                $ruleArray = array_filter($ruleArray, function($r) {
                    return !str_starts_with((string)$r, 'max:') && $r !== 'max';
                });
                $rules[$field] = array_values($ruleArray);
            }
        }

            // --- БАГ: negative_quantity ---
            if (Bug::isActive('negative_quantity')) {
                foreach ($rules as $field => $rule) {
                    if (str_contains($field, 'qty')) {
                        $ruleArray = is_array($rule) ? $rule : explode('|', $rule);
                        $ruleArray = array_filter($ruleArray, function($r) {
                            return !str_starts_with((string)$r, 'min:') && $r !== 'min';
                        });
                        $rules[$field] = array_values($ruleArray);
                    }
                }
            }

            return new \Illuminate\Validation\Validator(
                $translator, $data, $rules, $messages, $attributes
            );
        });
    }
}
```