Державний торговельно-економічний університет
Кафедра комп'ютерних наук та інформаційних систем

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
на тему:

«Розробка системи тестування Web-додатків»

Студента 4 курсу, 9 групи,
спеціальності
122 «Комп'ютерні науки»

_____
підпис студента

Гетьман
Антон
Артемович

Науковий керівник
кандидат фізико-математичних
наук

_____
підпис керівника

Філімонова
Тетяна
Олегівна

Гарант освітньої програми
кандидат технічних наук, доцент

_____
підпис керівника

Демідов
Павло
Георгійович

Київ 2023

Державний торговельно-економічний університет

Факультет інформаційних технологій
Кафедра комп'ютерних наук та інформаційних систем Спеціальність 122 «Комп'ютерні науки»

Затверджую
Зав. кафедри Пурський О.І.
«20» грудня 2022р.

Завдання
на випускну кваліфікаційну роботу студенту

Гетьман Антон Артемович
(прізвище, ім'я, по батькові)

- Тема випускної кваліфікаційної роботи

«Розробка системи тестування Web-додатків»
Затверджена наказом ректора від *«09» грудня 2022 р. № 3332*

- Строк здачі студентом закінченої роботи *30 травня2023 року*
- Цільова установка та вихідні дані до роботи

Мета роботи: розробка системи тестування Web-додатків.
Об'єкт дослідження: процес тестування Web-додатків.
Предмет дослідження: технології тестування Web-додатків.

- Перелік графічного матеріалу

- Консультанти по роботі із зазначенням розділів, за якими здійснюється консультування:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Філімонова Т.О.	15.12.2022 р.	15.12.2022 р.
2	Філімонова Т.О.	15.12.2022 р.	15.12.2022 р.
3	Філімонова Т.О.	15.12.2022 р.	15.12.2022 р.

- Зміст випускної кваліфікаційної роботи (перелік питань за кожним розділом)

ВСТУП
РОЗДІЛ 1 Аналітичне дослідження Web-додатків та рівнів тестування
1.1 Види Web-додатків, технології створення
1.2 Тестування Web-додатків
1.3 Рівні і види тестування програмного забезпечення

РОЗДІЛ 2 Організація розробки програмного засобу тестування Web-додатків
2.1 Сучасні підходи до тестування Web-додатків
2.2 Обґрунтування вибору програмних засобів для розробки системи тестування
2.3 Розробка моделі системи тестування Web-додатків

РОЗДІЛ 3 Реалізація розробки системи тестування Web-додатків
3.1 Програмна реалізація системи тестування
3.2 Апробація результатів дослідження
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

№ Пор.	Назва етапів випускної кваліфікаційної роботи	Строк виконання етапів роботи	
		За планом	фактично
1	2	3	4
1	*Вибір теми випускної кваліфікаційної роботи*	*04.10.2022*	*04.10.2022*
2	*Розробка та затвердження завдання на випускну кваліфікаційну роботу*	*15.12.2022*	*15.12.2022*
3	*Вступ*	*03.02.2023*	*03.02.2023*
4	*РОЗДІЛ 1 Аналітичне дослідження Web-додатків та рівнів тестування*	*28.02.2023*	*28.02.2023*
5	*РОЗДІЛ 2 Організація розробки програмного засобу тестування Web-додатків*	*06.04.2023*	*06.04.2023*
6	*РОЗДІЛ 3 Реалізація розробки системи тестування Web-додатків*	*12.05.2023*	*12.05.2023*
7	*Висновки*	*15.05.2023*	*15.05.2023*
8	*Здача випускної кваліфікаційної роботи на кафедру науковому керівнику*	*30.05.2023*	*30.05.2023*
9	*Попередній захист випускної кваліфікаційної роботи*	*31.05.2023 -01.06.2023*	*31.05.2023 -01.06.2023*
11	*Виправлення зауважень, зовнішнє рецензування випускної кваліфікаційної роботи*	*02.06.2023*	*02.06.2023*
12	*Представлення готової зшитої випускної кваліфікаційної роботи на кафедру*	*05.06.2023*	*05.06.2023*
13	*Публічний захист випускної кваліфікаційної роботи*	*За розкладом роботи ЕК*	

8. Дата видачі завдання «15» грудня 2022 р.

Керівник випускної кваліфікаційної роботи (проекту) Філімонова Т.О.
(прізвище, ініціали, підпис)

Гарант освітньої програми Демідов П.Г.
(прізвище, ініціали, підпис)

Завдання прийняв студент-дипломник Гетьман А.А.
(прізвище, ініціали, підпис)

9. Відгук керівника випускної кваліфікаційної роботи (проекту)

У випускній кваліфікаційній роботі розроблена система тестування Web-додатків на мові програмування C#.

Основними компонентами системи є інтеграційні тести, які тестують взаємодію елементів програми, коректність роботи системи та коректність виконання запитів до сервера, створено приклад додатку для тестування.

Випускна кваліфікаційна робота відповідає всім вимогам до випускних кваліфікаційних робіт. Всі поставлені завдання виконані. Випускна кваліфікаційна робота може бути допущена до захисту.

Керівник випускної кваліфікаційної роботи (проекту)

_____ 30.05.2023 р.

(підпис, дата)

10. Висновок про випускну кваліфікаційну роботу

Випускна кваліфікаційна робота студента _____
(прізвище, ініціали)

може бути допущена до захисту в екзаменаційній комісії.

Гарант освітньої програми Демідов П.Г.
(підпис, прізвище, ініціали)

Завідувач кафедри **Пурський О.І.**
(підпис, прізвище, ініціали)

«____»_____2023 р.

Анотація

Дана дипломна робота присвячена розробці системи тестування Web-додатків на мові програмування C#. Зростаюча популярність веб-додатків вимагає надійних і ефективних засобів для автоматичного тестування, що дозволить забезпечити якість і надійність цих додатків.

Основними компонентами розробленої системи є інтеграційні тести які тестують взаємодію елементів програми, коректність роботи системи та коректність виконання запитвів до сервера.

Досліджено теорію автоматизованого тестування додатків, проведено дослідження технологій програмної реалізації, створено приклад додатку для тестування та саму систему автоматизованого тестування.

Ключові слова: тестування веб-додатків, автоматизація тестування, C#, інструменти тестування**.**

Annotation

This paper is dedicated to the development of a system of testing of Web applications in the programming language C#. The growing popularity of web applications requires reliable and effective tools for automatic testing, which will ensure the quality and reliability of these applications.

The main components of the developed system are integration tests which test the interaction of the elements of the program, the correctness of the system and the correctity of the execution of requests to the server.

The theory of automated testing of applications was studied, research of software implementation technologies was conducted, an example of application for testing was created and the automated test system itself.

Keywords: web application testing, test automation, C#, test tools.

Зміст

ВСТУП

У сучасній розробці додатків, програм та веб-сервісів ключовим завданням є забезпечення якості продукту. Для досягнення цієї мети необхідне тестування, яке з розвитком технологій стає все більш складним і рутинним, тому автоматизоване тестування стає дедалі більш актуальним. Фреймворк автоматизованого тестування - це набір концепцій, інструментів та припущень, які забезпечують підтримку для автоматизованого тестування програмного забезпечення. Це дозволяє підвищити якість та конкурентоспроможність веб-сервісів.

Тестування - це процес виявлення помилок та дефектів у програмному забезпеченні, що може покращити якість продукту. Якість - це суб'єктивне поняття, і тестування може лише порівнювати стан і поведінку продукту зі специфікацією. Якість продукту зазвичай обмежується такими поняттями як коректність, надійність, практичність та безпечність.

Тестування не може гарантувати повну коректність роботи сторінки, але дозволяє порівнювати її стан і поведінку зі специфікацією та відповідними стандартами, такими як ISO 9126. IEEE 82-2088 Standard for Software Test Documentation[4] визначає склад та зміст супутньої документації процесу тестування.

Актуальність роботи. Розробка системи автоматизації тестування допоможе підвищити якість та конкурентоспроможність web-сервісів.

Мета роботи: розробка системи тестування Web-додатків.

Об’єкт дослідження: процес тестування Web-додатків.

Предмет дослідження: технології тестування Web-додатків.

Для досягнення поставленої мети необхідно виконати наступні **завдання**.

- Аналіз сучасних підходів до автоматизації тестування.
- Порівняння існуючих фреймворків для тестування.
- Розробка системи для автоматизації тестування.

Для вирішення поставлених задач використовувалися такі **методи**:

• загальнонауковий аналітичний метод;

• метод порівняльного аналізу для порівняння фреймворків для тестування;

• методи об’єктно-орієнтованого програмування для розробки додатку автоматизації тестування.

Практичне значення. Результати, отримані роботі, а саме, розроблений додаток, може бути використаний для автоматизації тестування web-сервісів.

Структура та обсяг випускної кваліфікаційної роботи. Випускна кваліфікаційна робота складається із вступу, трьох розділів, висновків, списку використаних джерел із 20 найменувань, і містить 59 сторінки основного тексту.

РОЗДІЛ 1
АНАЛІТИЧНЕ ДОСЛІДЖЕННЯ WEB-ДОДАТКІВ ТА РІВНІВ ТЕСТУВАННЯ

1.1 Види Web-додатків, технології створення

Web-додатки можна поділити за різними критеріями, такими як технології, які використовуються для їх розробки, способом
взаємодії з користувачем, функціональним призначенням тощо. Нижче наведено деякі загальні види Web-додатків:[19,20]

- Односторінкова інтерактивна програма відома як SPA(Single Page Application). Дуже важливо, щоб він був інтерактивним, як повноцінне програмне забезпечення, а не лише на одній сторінці. У результаті веб-сайт, який надає інформацію, може мати лише одну сторінку, але все одно не вважатися SPA. Односторінкова веб-програма дозволяє користувачеві переміщатися між вкладками, залишаючись на одній сторінці. Крім того, оновлюються та надсилаються лише основні частини вмісту, які сприяють швидкості SPA.

Gmail – це ілюстрація односторінкової програми. Зауважте, що адреса сторінки залишається незмінною, коли ви змінюєте списки розсилки. Цей SPA чудовий.

JavaScript є основною мовою, яку використовує SPA. Бібліотеку jQuery можна використовувати для створення простого односторінкового додатку.

- Статичні вебсайти складаються з вебсторінок, які відображаються без змін. Такі вебсторінки зазвичай призначені для надання інформації про компанію, її продукти чи послуги, контактну інформацію, години роботи, новини та інші відповідні дані, які будуть корисними для відвідувачів. Основною функцією таких сайтів є надання користувачам зручного та доступного джерела інформації, не передбачаючи взаємодії з користувачем, за винятком можливості перегляду та

зберігання цієї інформації.

Статичні сайти можуть бути створені з використанням простих HTML-редакторів, таких як Notepad++, або потужніших програм, таких як Dreamweaver або Adobe Muse. Крім того, можна використовувати готові шаблони для швидкого створення сайту.

Однак статичні сайти мають певні переваги, такі як низькі витрати на розробку та підтримку, висока швидкість і висока надійність, оскільки їхній вебкод не містить складних сценаріїв і баз даних. Крім того, такі сайти легко відображаються пошуковими системами, що дозволяє швидко і легко знаходити потрібну інформацію.

- Динамічні вебсайти є особливим типом веб-сайтів, які використовують програмне забезпечення для створення змінюваного вмісту на сервері. Для створення цих сторінок використовуються різні мови програмування, такі як PHP, Python, Ruby, JavaScript і т.д.

Однією з основних переваг динамічних веб-сторінок є можливість створювати динамічні та інтерактивні сторінки, які дозволяють користувачам взаємодіяти зі сторінкою. Наприклад, можливість надавати доступ користувачам до різних ресурсів, зберігання налаштувань та даних користувачів, автентифікації користувачів та інші подібні функції.

Динамічні сайти також можуть містити бази даних, що дозволяють зберігати, редагувати та оновлювати великі обсяги даних. Це може бути корисним для багатьох типів веб-сайтів, зокрема для сайтів електронної комерції, де необхідно зберігати велику кількість даних.

- MPA (Multi Page Application)[10]

Термін «електронна комерція» походить від англійського терміна «electronic commerce» і стосується будь-яких фінансових і комерційних операцій, які здійснюються через комп’ютерні мережі, а також пов’язаних з ними бізнес-процедур. Мобільна комерція, електронні грошові перекази, управління ланцюгом постачання,

обробка онлайн-транзакцій, електронний обмін даними (EDI), системи управління запасами та автоматизовані системи збору даних є основними елементами електронної комерції.

До електронної комерції відносять електронний обмін інформацією (англ. Electronis Data Interchange, EDI), електронний рух капіталу (англ. Electronic Funds Transfer, EFT), електронну торгівлю (англ. E-Trade), електронні гроші (E-Cash), електронний маркетинг (англ. E-Marketing), електронний банкінг (англ. E-Banking), електронні страхові послуги (англ. E-Insurance) тощо. Цей термін був придуманий і вперше використаний доктором Робертом Джейкобсоном, головним консультантом Комітету з питань комунального господарства та торгівлі штату Каліфорнія, в назві та тексті Закону про електронну комерцію в Каліфорнії.

E-commerce можна поділити на кілька категорій залежно від типу товару або послуги, яку продається[2]

- Електронна комерція для бізнесу, або B2B

Цей вид електронної комерції за своєю природою є бізнес-бізнесу. Наприклад, коли компанія пропонує свої продукти чи послуги іншій компанії, а не постійним клієнтам.

Яскравим прикладом є партнерство між Sony Corporation і Apple. Для iPhone і iPad Sony виробляє модулі камер і продає їх безпосередньо Apple. Подібно до цього, поки не перейшла на власний процесор Apple M1, Intel раніше постачала «яблучної» корпорації свої фірмові процесори.

- B2C – електронна комерція «бізнес-споживач».

B2C – це, мабуть, найпоширеніший вид електронної комерції, у якому компанія продає свої продукти, послуги та інші продукти безпосередньо клієнту. Apple пропонує власні товари, включаючи iPhone, iPad, iMac та інші товари.

- C2C означає електронну комерцію від споживача до споживача.

Коли клієнт продає продукт безпосередньо іншому споживачеві через Інтернет, навіть якщо він не виготовляв його сам, це відомо як C2C. Ви можете використовувати OLX (Україна)) як приклади для ілюстрації концепції електронної комерції. Користувачі можуть перерахувати тут практично будь-який товар для продажу, знайти покупців і продати. Природно, виробник в процедурі не бере участі.

- C2B означає електронну комерцію від споживача до бізнесу.

Насправді, це повна протилежність B2C, оскільки в цьому сценарії клієнт вже щось дає фірмі. Найпростішою ілюстрацією є те, що людина виробляє шкіряні вироби (гаманці, сумочки тощо) і продає їх як безпосередньо клієнтам, так і роздрібному інтернет-магазину.

- Ділове адміністрування (B2A)

Електронна комерція B2A стосується інтернет-транзакцій між підприємствами та державними органами. При роботі з різними юридичними документами, в тому числі, використовується B2A. Наприклад, тут можлива онлайн реєстрація на публічні закупівлі.

- C2A означає електронну комерцію від споживача до уряду.

Електронна комерція, що включає транзакції між клієнтами та державними установами, ще не набула великої популярності. Наприклад, запис до лікаря та оплата медичних послуг, оформлення податкової звітності та здійснення необхідних платежів, дистанційне навчання тощо.Веб сайти E-comerce мають багатий функціонал, такі як онлайн-платежі, знижки та промокоди, відгуки користувачів, аналітику відвідувань та продажів, та інші. Для успішної роботи e-commerce веб-сайту, важливо мати ефективну стратегію маркетингу, яка включає в себе SEO (пошукову оптимізацію), контекстну рекламу, email-маркетинг, та соціальні медіа.

- Сайти соціальних мереж належать до найпоширеніших категорій веб-програм. Ці програми дозволяють користувачам створювати власні профілі та ділитися різними типами вмісту з іншими користувачами, включаючи зображення,

відео, статті та інше. Соціальні мережі також можуть мати такі функції, як можливість підписатися на сайти інших користувачів, приватні повідомлення, групи обговорень тощо.

Веб-додатки соціальних медіа використовують різні технології та методи, щоб надати споживачам легкий доступ до інформації в безпечному середовищі. Зокрема, це може передбачати використання баз даних для зберігання даних про користувачів, їхні профілі та інші відомості, а також застосування алгоритмів для допомоги в організації потоку інформації та пропозицій. Соціальні мережі можуть бути як загальнодоступними, так і закритими, залежно від їхньої спрямованості. Наприклад, Facebook є загальнодоступним корпоративним веб-сайтом, і будь-хто може отримати доступ до нього для ведення бізнесу в Інтернеті. Так само, як і ви, LinkedIn є хорошим прикладом корпоративної політики, оскільки він був схвалений для використання та має доступ до власної енциклопедії.

Соціальні мережі мають здатність змінюватися та адаптуватися у відповідь на змінні вимоги клієнтів. Щоб зробити речі простішими та збільшити використання, такі додатки можуть містити нові функції. Бізнес-додатки – це ті, які дозволяють користувачам виконувати онлайн-транзакції, такі як замовлення товарів і послуг та ін.

1.2 Тестування Web-додатків

Тестування ПЗ - це процес перевірки правильності функціонування, та безпеки програмного забезпечення. Так як Web-додатки популяризуються , тестування стає все складнішим.[13,14]

Типи тестування ПЗ:

- Functional testing (тестування функціональності)
- Usability testing (тестування зручності використання)
- Interface testing (тестування інтерфейсу)

- Compatibility tasting (тестування на сумісність)
- Performance testing (тестування продуктивності)
- Security testing (тестування безпеки)

При функціональному тестуванні веб аплікацій тестується взагалі всі

об'єкти на веб сторінці, такі як: посилання, зв'язок до бази даних, різноманітні форми, допустиму форму реєстрації, форму логіна, форму зворотнього зв'язку і так далі. Де користувач вводить чи отримує інформацію від сторінки. Стосовно посилань , тестуються внутрішні посилання, які ведуть на інші сторінки цього ж сайту, зовнішні посилання і якщо є посилання з “email”, перевіряємо, що відкривається або ваш “email” клієнт або браузер. Далі ми тестуються також всі форми, які в нас є на сторінках. Це можуть бути форми реєстрації, відновлення паролю, форми зворотнього зв'язку і інше. Тестуються всі поля, перевіряється валідація полів.

Наступний вид тестування, який використовують це тестування

зручності. Перевіряються такі речі, як перевірка інтерфейсу, легкість у освоєнні сайту, легкість навігації, особиста субєктивна оцінка, задоволеності користування ресурсом і загальне враження. Від навігації показується , яким саме чином користувач переходить між сторінками, між пунктами меню, як користувач натискає, контролює посилання і таке інше. Зручність використання включає в себе такі поняття, як легкість користування, наявність головного меню на всіх сторінках сайту, послідовність, логічна структура у викладенні контенту і також, перевіряється сам контент. Контент повинен бути зрозумілий, логічний та структурований. Всі посилання повинні працювати. Також перевіряється інша інформація, яка буде корисна користувачеві, а саме мапа сайту.

Тут перевіряються три області: додаток, веб і сервер бази даних

Застосування: тестові запити правильно надсилаються до бази даних, а вихід на стороні клієнта відображається правильно. Помилки, якщо такі є, виявляються програмою та показується лише адміністратору, а не кінцевому користувачеві.

Веб-сервер: Тестовий веб-сервер обробляє всі запити програм без будь-якої відмови в обслуговуванні.

Сервер бази даних: перевірка що запити, надіслані до бази даних, дають очікувані результати. Перевірка реакції системи, коли зв’язок між трьома рівнями (додатком, Інтернетом і базою даних)не вдається встановити, а кінцевому користувачеві відображається відповідне повідомлення.

При тестуванні сумісності перевіряється, яким чином поведе себе додаток в різних середовищах. В тестування входить і сумісність по браузерах, кроссбраузерне тестування. Також як виглядає додаток на мобільному телефоні, у мобільному браузері, і також тестуються, як поводить себе система , чи не міняє вона свій вигляд, якщо ви використовуєте різні операційні системи.

Тестування продуктивності включає в себе такі перевірки, як час відгуку додатку при різній швидкості передачі даних, Тестування навантаження програми при нормальній та піковій нагрузці, тривале постійне навантаження і тестування проблем з пам'яттю, швидкодії процесора, робота з файлами і так далі., понаднормові навантаження те, що додаток не вийде з ладу. Якщо додаток не витримав нагрузки, то як швидко відновить свою роботу. Також перевіряються такі речі, як, наприклад, кеш, різноманітні стиснення даних, чіпування, що значно дозволяє покращити час завантаження об'єктів. Тестування продуктивності може перевіряти такі речі, як кількість юзерів за одиницю часу, пікові навантаження при стабільній кількості користувачів і навпаки, велика кількість даних, які передаються користувачем або користувачами одночасно.

Тестування безпеки. В першу чергу перевіряється, що всі дані користувачів є захищені. Доступ до цих даних мають лише авторизовані користувачі, і лише певне коло користувачів має доступ до, наприклад, завантаження файлів з обмеженими правами.

1.3 Рівні і види тестування програмного забезпечення.

Програми часто мають складну структуру, що складається з різних модулів та рівнів, і вони зазвичай працюють в кооперації з іншими додатками, комплексами та середовищами. Це дозволяє забезпечувати взаємодію між різними програмними продуктами та забезпечує їх ефективну роботу в цілому. Вищевказане призводить до виникнення різних рівнів тестування:[3,4]

1. Компонентне або модульне тестування перевіряє функціональність і шукає дефекти в частинах програми, які доступні і можуть бути протестовані окремо в силу своєї автономності (модулі програми, об'єкти, функції і т.д.). Ознакою, що утворює даний рівень, є те, що тестування окремих модулів можна виконувати на ранніх етапах розробки, поки робота над іншими модулями триває. Один з найбільш ефективних підходів до компонентного (модульного) тестування – це підготовка автоматизованих тестів до початку основного кодування ПЗ.

2. Інтеграційне тестування призначено для перевірки зв'язків між[7,18] компонентами програми, а також дослідження взаємодії додатку з середовищем, в рамках якого воно буде виконуватися.

3. Системне тестування направлено на дослідження функціональних і нефункціональних особливостей системи в цілому. При цьому виявляються дефекти, такі як невірне використання ресурсів системи, непередбачені комбінації даних рівня користувача, несумісність з оточенням, непередбачені сценарії використання, відсутня або невірна функціональність, незручність використання і т.д.[16]

4. Приймальне тестування проводиться на фінальному етапі розробки та спрямовано на з'ясування того, чи відповідає система приймально/здавальним критеріям до вимог замовника. Приймальне тестування виконується відповідно до плану приймальних робіт. Рішення про проведення приймального тестування приймається, коли: продукт досяг

необхідного рівня якості та замовник ознайомлений з Планом приймальних робіт або іншим документом, де описаний набір дій, пов’язаних з проведенням приймального тестування, дата проведення, відповідальні і т.д.

Розрізняють три основні форми тестування ПЗ:

Функціональні методи тестування зосереджені на аналізі функцій системи або її зовнішньої поведінки. До функціональних мтодів належать:

- функціональне тестування, яке зосереджується на тому, щоб система функціонувала належним чином і була присутня на всіх рівнях випробувань;
- тест безпеки, який перевіряє безпеку системи та оцінює ефективність стратегії запобігання несанкціонованому доступу та захисту конфіденційних даних;
- тест взаємодії, який оцінює ймовірність того, що додаток буде взаємодіяти з зовнішніми компонентами або системами, а також включає сумісність

Нефункціональні методи випробування, спрямовані на тестування кожного нефункціонального компонента системи. Ось деякі спецефічні тести для програмних характеристик:

- Тестування встановлення – направлене на перевірку запуску системи, а також на процеси видалення і оновлення ПЗ.
- Тестування зручності – оцінка зручності використання;
- Випробування на відмову та відновлення, що є частиною оцінки, що забезпечує надійність і безпеку системи;

оцінки продуктивності, які включають

3. Види тестів, пов'язаних зі змінами :

- Димове тестування, яке спрямоване на огляд кожної частини програми і виявлення основних недоліків, які можна побачити "Неозброєним Оком".
- Регресивне тестування відноситься до різних форм програмного забезпечення, призначених для виявлення дефектів у раніше протестоованих ділянках вихідного коду.

- Альфа-тестування, що імітує реальну роботу з розробниками стандартної системи або фактичну роботу з системою майбутніх Користувачів і Клієнтів
- Бета-тестування - за певних обставин розширення проводиться в попередній версії для більш широкої групи людей (часто з функціональними обмеженнями або робочим часом), щоб переконатися , що в кінцевому результаті є дуже мало проблем.[10]

У першій главі були розкриті питання про теорію тестування, які є методи та види , і у чому переваги.

З'ясували основи тестування , способи тестування, які можуть бути використані, і їх види, які були ретельно описані та організовані. Обговорили концепцію веб-тестування, особливості та сутність, Переваги та недоліки різних стратегій тестування. Типи запитів та поглиблений аналіз цих запитів також були характеризовані.

РОЗДІЛ 2
ОРГАНІЗАЦІЯ РОЗРОБКИ ПРОГРАМНОГО ЗАСОБУ ТЕСТУВАННЯ WEB-ДОДАТКІВ

2.1 Сучасні підходи до тестування Web-додатків

Сучасні методи тестування в C# допомагають програмістам виробляти

надійний, високоякісний код. Тестування в C# будується на принципі одиничного тестування. Це дозволяє програмістам перевіряти, що конкретні методи, класи або елементи програми є правильними і працюють так, як було призначено. Фреймворки NUnit, xUnit і Microsoft MSTest є популярними виборами для створення одиничних тестів в C#.[4]

Деякі з них наступні:

Тестування інтеграції: спрямовані на те, щоб з'ясувати, як різні частини програми взаємодіють, щоб виявити будь-які потенційні проблеми або несправності під час інтеграції модулів, баз даних або зовнішніх послуг. Можливе створення тестів інтеграції в C#, використовуючи рамки, такі як NUnit, xUnit або SpecFlow.

1. Рамка NUnit [7]пропонує ряд класів і атрибутів для полегшення створення та виконання тестів. Під час написання тестів інтеграції в NUnit можна відзначити тест-клас спеціальним атрибутом [TestFixture] і індивідуальні методи тестування атрибут [TEST]. За допомогою атрибутів [SetUp] і [TearDown], NUnit також дозволяє виконувати конкретні операції налаштування або відключення перед або після тестування.

2. xUnit[17]: Ця рамка також дозволяє писати тести інтеграції C#. Методи тестування в xUnit ідентифікуються атрибутом [Fact]. Подібно до NUnit, атрибути [Setup] і [Teardown] дозволяють використовувати певні налаштування або дії teardown. xUnit додатково пропонує атрибіти [Trait] і (TraitDiscoverer), які можна використовувати для групового тестування.

3. Рамка тестування під назвою SpecFlow дозволяє писати тести в режимі поведінкового розвитку (BDD). BDD - це метод розробки програмного забезпечення, який робить сильний акцент на бізнес-аналітиків, тестувачів і розробників, які мають спільне розуміння вимог і поведінки системи.

Функціональні випробування перевіряють, що програма працює відповідно до вимог і специфікацій. Він перевіряє, що взаємодії з програмою дають очікувані результати і дії. Найчастіше використовувані рамки для створення функціональних тестів в C# - це Selenium або SpecFlow.[9]

Випробування mock: тестування mockпередбачає створення об'єктів(mocktesting), які імітують реальні об'єкти. Це дозволяє програмістам перевірити, чи їхній код взаємодіє і правильно обробляє поведінку цих об'єктів. При написанні помилкових тестів в C# часто використовуються рамки, такі як Moq або

NSubstitute.

Створення штучних об'єктів для перевірки поведінки реального об'єкта, з яким взаємодіє тестуваний код, є поширеною практикою у тестуванні програмного забезпечення. Mocks можуть імітувати різні сценарії або демонструвати конкретну поведінку, як це визначено програмістом. Вони також можуть дати передбачувані результати.

Перевірка того, як тестуваний код взаємодіє з його залежностями, такими як сервіси, бази даних або зовнішні API, є основною метою тестування помилок. Відділяючи тестуваний код від його залежностей, він дозволяє перевірити саму логіку.

Для впровадження тестів зображення в C# часто використовуються рамки, такі як Moq і NSubstitute. Ці рамки пропонують прості способи змалювати об'єкти, вказати, як вони повинні поводитися, і підтвердити взаємодію з ними.

Процес MOck тестування:

1. Mock object creation: Синтетичний об'єкт, який імітує поведінку реального об’єкта, створюється за допомогою рамки mock.

2. Налаштування очікуваних значень, повернення значень та будь-яка інша необхідна поведінка макарону полегшить тестування.

3. Завантажується код під час тестування: Тестований код запускається і взаємодіє з об'єктом.

4. Перевірка взаємодій мак: за допомогою функцій рамки мак визначається, чи дійсно відбулися очікувані взаємодії, включаючи виклики методу та пройшли аргументи, з об'єктом мак.

5. Перевірка результатів: підтверджується точність результатів, отриманих перевіреним кодом.

Оскільки мок об'єкти не потребують реальних залежностей і їх легко налаштувати для різних сценаріїв, вони роблять тестування більш ефективними і швидкими. Крім того, вони дають тестовому середовищу більше контролю і

допомагають запобігти взаємозалежності тестів.

Випробування продуктивності та навантаження: ці тести дивляться на те, наскільки добре програма виконує і обробляє завантаження. Вони допомагають знайти проблеми зі швидкістю, використанням пам'яті або продуктивністю, коли є багато навантаження. Для створення тестів навантаження та продуктивності в C# використовуйте рамки, такі як BenchmarkDotNet або Apache JMeter.

Різні рамки та інструменти можуть бути використані для створення тестів продуктивності та навантаження в C#. Ось кілька із них:

1. Рамка BenchmarkDotNet пропонує потужні інструменти для написання та виконання тестів продуктивності C#. Він дозволяє контролювати різні показники продуктивності, включаючи час виконання, використання пам'яті, швидкість роботи та багато іншого. BenchmarkDotNet надає зручний синтаксис для створення тестів, а результати звітів генеруються автоматично.

2. Apache JMeter[15] є відомим інструментом тестування продуктивності та навантаження з відкритим кодом. Він дозволяє моделювати різні сценарії навантаження, симулювати фактичне використання додатків та оцінювати продуктивність системи. Веб-додатки, служби, бази даних та інші компоненти програмного забезпечення можуть бути протестовані за допомогою Apache JMeter.

3. Тестування продуктивності можливе завдяки вбудованим функціоналам у Visual Studio, якщо ви використовуєте його. Ви можете розробити тести для оцінки продуктивності вашого додатка під час симуляції навантаження. Можливості тестування навантаження Visual Studio включають можливість збирати метрики, вивчати результати тестів та інтегруватися з іншими інструментами розробки.

Ці рамки та інструменти дозволяють ретельно перевіряти продуктивність та навантаження програмного забезпечення, що дозволяє виявляти проблеми та покращувати його ефективність у реальному використанні.

Ці методи тестування гарантують точність, надійність та ефективність C#-коду.

До сучасних технології та підходів до тестування відносяться : розробка через тестування (англ. Test Drive Development (TDD) і розробка на основі поведінки (англ. Behavior Driven Development,

BDD).

Техніка розробки програмного забезпечення, відома як «Розвиток за допомогою тестування» (TDD), починається з початкового написання тестів, після чого код записується для проходження тестів. Після цього здійснюється рефакторизація (підвищення якості) коду до визнаного стандарту.

Behavioral Driven Development (BDD) - це варіація розробки через тестування, яка дозволяє розробляти тестові випадки та створювати краще програмне забезпечення при тестуванні додатків відповідно до очікувань клієнтів. Основна мета BDD полягає в дослідженні завдань, які формулюються за допомогою слів і тестів на основі цих завдань, а також логіки .

Процес розробки проходить у три етапи

- Discovery: дослідження проблеми клієнта та можливість їївирішення
- Formulation: формулювання прийнятих задач у вигляді бізнес-специфікації та
- Automation: автоматизація – створення відповідних автоматизованих тестів.

Кожна з цих технологій в автоматизованому тестуванні має свої переваги і недоліки.

Захист програмного забезпечення. Застосування методології TDD для одиничного тестування або підтвердження того, що функції Separate працюють, має сенс. Згодом, як BDD - для тестування інтеграції, тобто для випробування того, як різні модулі взаємодіють один з одним, що тягне за собою тести різних користувачів.

2.2 Обґрунтування вибору програмних засобів для розробки системи тестування

Для створення тестів була обрана мова C#.

C# була створена Microsoft і також є однією з домінуючих мов в автоматизації. Це одна з найбільш широко використовуваних мов і дотримується OOP principles.Net Framework. У опитуванні розробників Stack Overflow (2019) 67% респондентів оцінили C# як найкращу мову для автоматизації, програмування та ін. Додатки, що працюють на Windows, Android та iOS, добре підходять для автоматизованого тестування за допомогою C#.

Ця мова розвитку поступово, але впевнено набирає імпульсу в області автоматизованого тестування. Багато розробників автоматизації віддають перевагу використанню C# для тестування автоматизації та перехресних тестів через міцні можливості мови та сумісність з Selenium WebDriver. [9]

Індустрія автоматизованих випробувань поступово, але впевнено набирає форму з цією мовою розвитку. Тестувачі можуть створювати ефективний тест-код, який легко підтримувати, використовуючи модель дизайну об'єкта сторінки (POM).

Якщо йдеться про тестування веб-додатків, C# є однією з кращих мов програмування для цієї мети. Ось кілька причин, чому обирають C# для тестування веб-додатків:

Надійність та безпека: C# є мовою програмування, що компілюється, що дозволяє перевірити код на ранній стадії. Більшість помилок можна виявити під час компіляції, що забезпечує більшу надійність програми. Також, C# має вбудовані механізми безпеки, що дозволяє зменшити ризик вразливостей в програмі.

Зручний для написання автоматизованих тестів: C# має чітке синтаксичне побудова, що дозволяє розробникам з легкістю створювати автоматизовані тести.

Зокрема, фреймворк NUnit, що розроблений на C#, дозволяє зручно тестувати веб-додатки.

Широкі можливості для розробки: C# має доступ до багатьох бібліотек та фреймворків, що дозволяють легко розширювати можливості програми. Зокрема, для тестування веб-додатків можна використовувати бібліотеки, такі як Selenium, які мають великий функціонал та допомагають легко тестувати веб-інтерфейс.

Підтримка від Microsoft: C# є мовою програмування, що розробляється та підтримується компанією Microsoft, тому розробникам не доведеться стикається з проблемами з підтримкою мови або фреймворків. Отже, враховуючи всі переваги, C# є хорошим вибором для тестування веб-додатків. Однією з найвідоміших платформ розробки веб-додатків Microsoft є ASP.NET. Він забезпечує високий рівень зручності та багато переваг[11,12,16]. Деякі з них наступні:

1. Розширюваність: ASP.NET - це потужна рамка розробки веб-додатків, яка надає можливість розширювати та масштабувати додатки для обробки великого обсягу одночасних користувачів та запитів.За допомогою кластерів серверів ASP.NET може розподілити навантаження між кількома серверами. Це покращує масштабність та високу доступність програми. Групування серверів забезпечує гнучкість і можливість збільшення потужності системи, дозволяючи додавати або видаляти сервери відповідно до вимог програми.

Крім того, ASP.NET дозволяє горизонтальне масштабування, що дозволяє додавати більше серверів, щоб рівномірно розподілити навантаження і гарантувати найвищу продуктивність. Це дозволяє збільшити масштаби обчислювальних ресурсів для вирішення вимог програми або додати більше серверів для обробки збільшеного трафіку.

2. Щоб захистити веб-додатки, ASP.NET пропонує сильні механізми безпеки. Щоб зупинити атаки та порушення, він має механізми валідації входів, контролю

доступу, аутентифікації та авторизації.

3. ASP.NET забезпечує високопродуктивні веб-додатки за допомогою інтегрованих можливостей кешування та компіляції. Він використовує модель обробки запитів, яка була оптимізована для ефективності сервера.

4. Розробка спрощена завдяки великому набору інструментів і бібліотек, які надає ASP.NET. Він підтримує C# і VB.NET, а також інші мови програмування, і пропонує широкий спектр попередньо побудованих компонентів для створення функціональності додатків.

5. Інтеграція: SQL Server, Azure, Active Directory та інші продукти Microsoft бездоганно інтегровані з ASP.NET. Це дозволяє збільшити функціональність веб-додатків і полегшити їх взаємодію з іншими системами.

6. ASP.NET пропонує різноманітні варіанти інтеграції для продуктів Microsoft. Наприклад, система управління реляційними базами даних SQL Server тісно інтегрована з ASP.NET. Це дозволяє використовувати мову SQL для запитування баз даних та інших корисних операцій бази даних, таких як читання та написання даних. Крім того, ASP.NET і хмарна платформа Microsoft Azure працюють разом без будь-яких проблем. Це дозволяє завантажувати веб-додатки на Azure та використовувати різні хмарні послуги, включаючи бази даних, кешування, зберігання файлів, інтеграцію послуг штучного інтелекту та багато іншого. Веб-додатки мають гнучкість, масштабуваність та надійність завдяки інтеграції з Azure.

1. Підтримка: Для того, щоб розробникам було простіше отримати допомогу та знайти рішення, Microsoft пропонує ASP.NET з великою кількістю документації, ресурсів та підтримки спільноти.

Microsoft створила рамку для розробки веб-сайтів під назвою ASP.NET. Ця рамка пропонує найсучаснішу підтримку для створення надійних веб-сторінок, веб-служб та додатків. Однак під час розробки програмного забезпечення можуть виникнути проблеми та запитання, і в цих ситуаціях важливо мати доступ до ресурсів

та підтримки, щоб знайти відповіді якомога швидше.

Розробники, які використовують ASP.NET, мають доступ до великої кількості документації та ресурсів від Microsoft. Керівництва, підручники, зразки коду та пропозиції щодо розробки цієї структури включені до офіційної документації. Ви будете направлятися крок за кроком через різні функції ASP.NET за допомогою цієї професійної документації Microsoft.

Крім офіційної документації, ви також можете скористатися ресурсами спільноти. Існує велика спільнота розробників, які працюють з ASP.NET, і ви можете знайти форуми, блоги, веб-семінари та інші джерела, де ви можете отримати відповіді на свої питання. Багато розробників також діляться своїм досвідом і знаннями, надаючи корисні поради та рішення проблем.

Тому корпорація Майкрософт створила умови для того, щоб розробникам було простіше отримати допомогу і знайти рішення при роботі з ASP.NET. Багатство документації, ресурсів та підтримки спільноти надає розробникам необхідні інструменти для успішного розробки веб-додатків. Завдяки цим функціям ASP.NET часто використовується для створення надійних, безпечних та масштабованих веб-додатків.

2.3 Розробка моделі системи тестування Web-додатків

Розробка моделі системи тестування Web-додатків включає створення структури та компонентів, що дозволять проводити автоматизоване тестування різних аспектів Web-додатків. Основна мета моделі - забезпечити надійне та ефективне тестування функціональності, взаємодії та коректності Web-додатків перед їх випуском.

Основні компоненти моделі системи тестування Web-додатків можуть включати наступні:

Частина 1: Шкільний API

Компонент Школа, представляє з себе репрезентацію звичайного шкільного закладу у вигляді API, що дозволяє користувачам взаємодіяти з його основними функція і так далі. Ось основні компоненти API шкільної програми:

Шкільний контролер: керує операціями, пов'язаними з управлінням шкільними суб'єктами, такими як створення нових шкіл, отримання шкільних даних, оновлення шкільної інформації та видалення шкіл.

Студентські команди: надає команди або кінцеві точки для створення, отримання, оновлення та видалення студентських суб'єктів. Ці команди викликаються через API для шкільних додатків, але можуть бути реалізовані в окремих компонентах.

Команди вчителя: надає команди або кінцеві пункти для управління навчальними суб'єктами, включаючи створення, пошук, оновлення та видалення. Аналогічно, ці команди викликаються через API шкільної програми, але можуть бути реалізовані в окремих компонентах.

Команди кімнати: пропонує команди або кінцеві точки для обробки кімнатних суб'єктів, таких як створення, пошук, оновлення та видалення кімнат. Як і попередні суб'єкти, ці команди закликаються через API для шкільних додатків, але можуть бути реалізовані в окремих компонентах.

Автентифікація та авторизація API: впроваджує механізми автентизації для захисту кінцевих точок API та обмеження доступу до авторизованих користувачів. Він може використовувати такі методи, як API-ключі, токени або OAuth для аутентифікації та управління доступом на основі ролей (RBAC) для авторизації.

Частина 2: База даних

Компонент бази даних зберігає дані, що генеруються та використовуються API шкільних додатків. Він забезпечує постійне зберігання та отримання інформації, пов'язаної зі школами, студентами, вчителями та кімнатами. Основними елементами бази даних є:

Шкільна таблиця: зберігає шкільні записи з атрибутами, такими як ID школи,

ім'я, адреса, номер.

Студентський стіл: зберігає записи студентів з атрибутами, такими як ідентифікатор студента, ім'я, прізвисько, кімната,школа віктощо.

Таблиця вчителя: зберігає записи вчителя з атрибутами, такими як ідентифікатор вчителя, ім'я, предмет, контактні дані тощо.

Таблиця кімнат: зберігає записи кімнат з атрибутами, такими як ідентифікатор кімнати, номер кімнат, урок, школа.

Система управління базами даних: Використовує систему управління реляційною базою даних (RDBMS) як MySQL, PostgreSQL або Oracle для управління базою і обробки операцій, таких як введення даних, пошук, оновлення та видалення.

Частина 3: Система тестування

Компонент системи тестування відповідає за проведення тестів на API Школи, в тому числі тестів інтеграції. Він перевіряє функціональність, продуктивність та надійність API і гарантує, що він відповідає очікуваним вимогам. Основними елементами компонентів системи тестування є:

Визначення випадків тестування: визначає окремі випадки тестувань, що охоплюють різні сценарії та функціональності API для шкільних додатків. Кожен випробувальний випадок визначає входи, очікувані виходи та кроки, які потрібно виконати.

Двигун виконання тестів: Виконує визначені випадки тестування і захоплює фактичні результати, порівнюючи їх з очікуваними результатами.

Тестування інтеграції: зосереджується на тестуванні взаємодій та потоку даних між контролером школи та командами для студентів, викладачів та суб'єктів кімнати. Це забезпечує належну інтеграцію і функціональність.

Звітність та аналіз: генерує всеосяжні звіти про випробування, які дають уявлення про результати тестів, включаючи пройшли тести, невдалі тести, охоплення тестів та будь-які виявлені проблеми або недоліки.

Автоматизація: підтримує автоматизацію виконання тестів, що дозволяє автоматизувати повторні тести для ефективності та послідовності.

Інтегруючи систему тестування з API для шкільних додатків, можете періодично виконувати тести, щоб забезпечити правильне функціонування додатків і підтримувати очікуваний рівень якості.

РОЗДІЛ 3

Реалізація розробки системи тестування Web-додатків

3.1 Програмна реалізація системи тестування

Написання інтеграційних тестів, які перевірятимуть роботу веб-додатка з базою даних MySQL. Використання фреймворку для тестування, наприклад, NUnit або для створення і запуску тестових сценаріїв.

Перед запуском тестів створюються підготовчі дані в базі даних MySQL. Заповнюються таблиці необхідними даними для виконання тестових сценаріїв.

Інтеграційні тести, перевірятимуть різні аспекти роботи веб-додатка з базою даних. Наприклад чи виконуються запити до бази даних правильно, чи повертаються очікувані результати, чи відбувається коректне управління транзакціями тощо.

Очищення тестових даних: Після виконання тестів здійснюється очищення тестових даних з бази даних.

1. CreateSchool

```
public static class CreateSchool
{
public sealed record Command(string Name, string Number, string Address) : IRequest<Unit>;
public sealed class Handler : IRequestHandler<Command, Unit>
{
private readonly TestDbContext _dbContext;
        public Handler(TestDbContext dbContext)
        {
            _dbContext = dbContext;
        }
```

```
        public async Task<Unit> Handle(Command request, CancellationToken cancellationToken)
{
            var newSchool = School.Create(request.Name,
            request.Number,
            request.Address)
            await _dbContext.Set<School>()
               .AddAsync(newSchool);
            await _dbContext.SaveChangesAsync();
            return Unit.Value;
        }
}
}
```

Цей код визначає статичний клас “CreateSchool”, який містить команду “Command” та обробник “Handler” для створення нової школи.

Команда “Command” містить інформацію про нову школу, таку як ім'я, номер та адресу.

Обробник “Handler” відповідає за обробку команди “Command” і повернення результату. У методі “Handle” обробляється команда створення школи.

Для цього створюється новий об'єкт школи з використанням отриманих даних про ім'я, номер та адресу школи.

Після цього нова школа додається до контексту бази даних, щоб зберегти її дані в базі.

Після збереження викликається метод “SaveChangesAsync”, який зберігає зміни у базі даних.

На кінці метод повертає значення “Unit.Value”, яке вказує на успішне виконання команди створення школи.

Отже, цей код реалізує логіку створення нової школи, збереження її даних в базі даних

і повернення підтвердження про успішне створення.

1.1. CreateSchool_ReturnsSchool_WhenSchoolCreatedSuccessfully

```
{
    string name = $"New school-{Guid.NewGuid()}";
    _ = await _mediator.Send(new CreateSchool.Command(name, "22", "Test"));
    var newSchool = await _dbContext.Schools
       .FirstOrDefaultAsync(c => c.Name == name);
    Assert.NotNull(newSchool);
    Assert.Equal(name, newSchool.Name);
}
```

Цей тест перевіряє, чи успішно створюється школа та чи відбувається коректне збереження цих даних в базі даних.

2. CreateRoom

```
public static class CreateRoom
{
public sealed record Command(Guid SchoolId, string Number, string Lesson) : IRequest<Unit>;
public sealed class Handler : IRequestHandler<Command, Unit>
{
private readonly TestDbContext _dbContext;
    public Handler(TestDbContext dbContext)
    {
      _dbContext = dbContext;
    }
    public async Task<Unit> Handle(Command request, CancellationToken cancellationToken)
```

```
{
        var currentSchool = await _dbContext.Schools
        .FirstOrDefaultAsync(c => c.Id == request.SchoolId);
        if (currentSchool is null)
        {
           throw new ArgumentNullException("School not found at database");
        }
        var newRoom = Room.Create(request.Number,
          request.Lesson);
        currentSchool.AddRoom(newRoom);
        await _dbContext.SaveChangesAsync();
        return Unit.Value;
    }
}
}
```

Цей код представляє собою визначення статичного класу “CreateRoom”, який містить команду “Command” та обробник “Handler” для створення нової кімнати в школі. Команда “Command” приймає параметри, такі як ідентифікатор школи “(SchoolId)”, номер кімнати “(Number)” та урок, який проводиться в цій кімнаті “(Lesson)”.

Обробник “Handler” відповідає за обробку команди “Command” і повернення результату. У методі “Handle” обробляється команда створення кімнати.

Спочатку отримується поточна школа з бази даних за допомогою ідентифікатора “SchoolId”. Якщо школа не знайдена “(currentSchool is null)”, викидається виняток “(ArgumentNullException)”, що повідомляє про те, що школу не знайдено в базі даних.

Далі створюється новий об'єкт кімнати з використанням отриманих даних про номер кімнати та урок.

Потім нова кімната додається до списку кімнат школи

“(currentSchool.AddRoom(newRoom))”.

Зміни зберігаються в базі даних за допомогою методу “SaveChangesAsync”.

На кінці метод повертає значення “Unit.Value”, що вказує на успішне виконання команди створення кімнати.

Отже, цей код реалізує логіку створення нової кімнати в школі, збереження її даних в базі даних і повернення підтвердження про успішне створення

2.1. CreateRoom_ReturnsRoom_WhenRoomCreateSuccessfully

```
{
    string name = $"New school-{Guid.NewGuid()}";
    _ = await _mediator.Send(new CreateSchool.Command(name, "22", "Test"));

    var currentSchoolId = await _dbContext.Schools
      .Where(c => c.Name == name)
      .Select(c => c.Id)
      .FirstOrDefaultAsync();
    string number = $"{Guid.NewGuid()}";
    _ = await _mediator.Send(new CreateRoom.Command(currentSchoolId,
      number,
      "Test"));
    var currentSchool = await _dbContext.Schools
      .FirstOrDefaultAsync(c => c.Id == currentSchoolId);
    var currentRoom = currentSchool!
      .Rooms
      .FirstOrDefault(c => c.Number == number);
    Assert.NotNull(currentRoom);
    Assert.Equal(number, currentRoom.Number);
```

```
    }
```

Спочатку створюється нова школа з випадково згенерованим ім'ям за допомогою команди “_mediator.Send(new CreateSchool.Command(name, "22", "Test"))”. Після створення школи отримується її ідентифікатор “(currentSchoolId)”, використовуючи LINQ запит до бази даних. Далі створюється нове приміщення (кімната) в цій школі за допомогою команди “_mediator.Send(new CreateRoom.Command(currentSchoolId, number, "Test"))”.

Після створення кімнати знову отримується посилання на поточну школу з бази даних “(currentSchool)”. З цієї школи отримується перше приміщення (кімната) з заданим номером “(number)” за допомогою LINQ запиту: “currentSchool!.Rooms.FirstOrDefault(c => c.Number == number)”.

2.2. CreateRoom_ThrowsException_WhenSchoolNotExists

```
{
        string number = "123";
        await Assert.ThrowsAsync<ArgumentNullException>(() => _mediator.Send(new
CreateRoom.Command(Guid.NewGuid(),
            number,
            "Test")));
    }
```

Перевіряється, чи викидається виняток, коли є спроба створити кімнату в неіснуючій школі. Він має наступну структуру:

Задається фіксований номер кімнати “(number = "123")”.

Викликається команда “CreateRoom.Command” з випадковим ідентифікатором школи (якої не існує).

Використовується метод “Assert.ThrowsAsync” для перевірки того, що виняток “ArgumentNullException” викидається при спробі створити кімнату в неіснуючій школі.

Цей тест перевіряє, чи правильно обробляється випадок, коли намагаємося створити кімнату в школі, яка не існує, і чи викидається очікуваний виняток.

3. CreateStudent

```
public static class CreateStudent
{
public sealed record Command(Guid SchoolId, Guid RoomId, string FirstName, string LastName, int Age) : IRequest<Unit>;

public sealed class Handler : IRequestHandler<Command, Unit>
{
private readonly TestDbContext _dbContext;
        public Handler(TestDbContext dbContext)
        {
            _dbContext = dbContext;
        }

        public async Task<Unit> Handle(Command request, CancellationToken cancellationToken)
{
            var currentSchool = await _dbContext.Schools
            .FirstOrDefaultAsync(c => c.Id == request.SchoolId);
            if (currentSchool is null)
            {
                throw new ArgumentNullException("School not found at database");
            }
            var currentRoom = currentSchool
                .Rooms
```

```
            .FirstOrDefault(c => c.Id == request.RoomId);
        if (currentRoom is null)
        {
            throw new ArgumentNullException("Room not found at database");
        }
        var newStudent = Student.Create(request.FirstName,
            request.LastName,
            request.Age);
        currentRoom.AddStudent(newStudent);
        await _dbContext.SaveChangesAsync();
        return Unit.Value;
    }
}
}
```

Цей код представляє собою статичний клас “CreateStudent”, який містить команду “Command” та обробник “Handler” для створення нового студента в кімнаті школи.

Команда “Command” містить параметри для створення студента, такі як ідентифікатор школи “(SchoolId)”, ідентифікатор кімнати “(RoomId)”, ім'я “(FirstName)”, прізвище “(LastName)” та вік “(Age)” студента.

Обробник “Handler” відповідає за обробку команди “Command” і повернення результату. У методі “Handle” обробляється команда створення студента.

Спочатку отримується поточна школа з бази даних за допомогою ідентифікатора "SchoolId”. Якщо школа не знайдена “(currentSchool is null)”, викидається виняток “(ArgumentNullException)”, що повідомляє про те, що школу не знайдено в базі даних.

Далі отримується поточна кімната школи за її ідентифікатором “RoomId”. Якщо кімната не знайдена “(currentRoom is null)”, викидається виняток

“(ArgumentNullException)”, що повідомляє про те, що кімнату не знайдено в базі даних.
Створюється новий об'єкт студента з отриманими даними про ім'я, прізвище та вік.
Студент додається до списку студентів кімнати “(currentRoom.AddStudent(newStudent))”.
Зміни зберігаються в базі даних за допомогою методу “SaveChangesAsync”.
На кінці метод повертає значення “Unit.Value”, що вказує на успішне виконання команди створення студента в кімнаті школи.
Отже, цей код реалізує логіку створення нового студента в певній кімнаті школи, збереження його даних в базі даних і повернення підтвердження про успішне створення.

3.1. CreateStudent_ReturnsStudent_WhenStudentCreatedSuccessfully

```
{
    string name = $"New school- {Guid.NewGuid()}";
    _ = await _mediator.Send(new CreateSchool.Command(name, Guid.NewGuid().ToString(), "Test"));
    var currentSchoolId = await _dbContext.Schools
      .Where(c => c.Name == name)
      .Select(c => c.Id)
      .FirstOrDefaultAsync();
    string number = Guid.NewGuid().ToString();
    _ = await _mediator.Send(new CreateRoom.Command(currentSchoolId,
      number,
      "Test"));
    var currentSchool = await _dbContext.Schools
      .FirstOrDefaultAsync(c => c.Id == currentSchoolId);
```

```
    var currentRoom = currentSchool!
        .Rooms
        .FirstOrDefault(c => c.Number == number);
    string firstName = $"Test-{Guid.NewGuid()}";
    await _mediator.Send(new CreateStudent.Command(currentSchoolId,
        currentRoom!.Id,
        firstName,
        "Test",
        21));
    currentSchool = await _dbContext.Schools
        .FirstOrDefaultAsync(c => c.Id == currentSchoolId);
    currentRoom = currentSchool!
        .Rooms
        .FirstOrDefault(c => c.Number == number);
    var newTeacher = currentRoom!
        .Students
        .FirstOrDefault(c => c.FirstName == firstName);
    Assert.NotNull(newTeacher);
    Assert.Equal(firstName, newTeacher.FirstName);
}
```

Основні кроки в цьому тесті наступні:

- Отримується ідентифікатор поточної школи, шляхом пошуку в базі даних школи з відповідним ім'ям за допомогою “await _dbContext.Schools.Where(c => c.Name == name).Select(c => c.Id).FirstOrDefaultAsync()”;.
- Отримується поточна школа з бази даних за допомогою ідентифікатора школи.
- Отримується поточна кімната з поточної школи за номером кімнати.

- Створюється новий студент з унікальним ім'ям, прізвищем і віком за допомогою команди “CreateStudent.Command”.
- Зберігаються зміни в базі даних за допомогою “await _dbContext.SaveChangesAsync()”;.
- Отримується оновлена поточна школа з бази даних.
- Отримується оновлена поточна кімната з поточної школи.
- Отримується новий студент з поточної кімнати за його іменем.
- Перевіряється, чи не є отриманий студент “null” і чи співпадає його ім'я з очікуваним.

Таким чином, цей тест перевіряє, чи відбувається успішне створення студента в кімнаті школи і чи повертається цей студент після створення.

4. CreateTeacher

```
public static class CreateTeacher
{
public sealed record Command(Guid SchoolId, Guid RoomId, string FirstName, string LastName, int Age) : IRequest<Unit>;
public sealed class Handler : IRequestHandler<Command, Unit>
{
private readonly TestDbContext _dbContext;
        public Handler(TestDbContext dbContext)
        {
            _dbContext = dbContext;
        }
        public async Task<Unit> Handle(Command request, CancellationToken cancellationToken)
{
```

```
        var currentSchool = await _dbContext.Schools
        .FirstOrDefaultAsync(c => c.Id == request.SchoolId);
        if (currentSchool is null)
        {
            throw new ArgumentNullException("School not found at database");
        }
        var currentRoom = currentSchool
            .Rooms
            .FirstOrDefault(c => c.Id == request.RoomId);
        if (currentRoom is null)
        {
            throw new ArgumentNullException("Room not found at database");
        }

        var newTeacher = Teacher.Create(request.FirstName,
            request.LastName,
            request.Age);
        currentRoom.AddTeacher(newTeacher);
        await _dbContext.SaveChangesAsync();
        return Unit.Value;
    }
}
}
```

В цьому коді визначено клас “CreateTeacher”, який містить команду “Command” і обробник “Handler” для створення нового вчителя в кімнаті школи.

У команді “Command” передаються параметри для створення нового вчителя, такі як

ідентифікатор школи, ідентифікатор кімнати, ім'я, прізвище та вік вчителя.

В обробнику “Handler” отримуються посилання на контекст бази даних із залежності “TestDbContext”. У методі “Handle” обробляється команда створення вчителя.

Спочатку здійснюється пошук школи в базі даних за її ідентифікатором. Якщо школа не знайдена, виникає виняток “ArgumentNullException”.

Потім здійснюється пошук кімнати в поточній школі за її ідентифікатором. Якщо кімната не знайдена, виникає виняток “ArgumentNullException”.

Далі створюється новий вчитель за допомогою фабричного методу “Teacher.Create”, який приймає параметри ім'я, прізвище та вік вчителя.

Новий вчитель додається до кімнати за допомогою методу “AddTeacher”, і зміни зберігаються в базі даних за допомогою “SaveChangesAsync”.

Нарешті, повертається об'єкт “Unit”, що позначає успішне виконання команди створення вчителя.

Отже, цей код виконує процес створення нового вчителя в певній кімнаті школи і зберігає ці зміни в базі даних.

4.1. CreateTeacher_ReturnsTeacher_WhenTeacherCreatedSuccessfully

```
{
    string name = $"New school- {Guid.NewGuid()}";
    _ = await _mediator.Send(new CreateSchool.Command(name,
Guid.NewGuid().ToString(), "Test"));
    var currentSchoolId = await _dbContext.Schools
      .Where(c => c.Name == name)
      .Select(c => c.Id)
      .FirstOrDefaultAsync();
    string number = Guid.NewGuid().ToString();
```

```
_ = await _mediator.Send(new CreateRoom.Command(currentSchoolId,
    number,
    "Test"));
var currentSchool = await _dbContext.Schools
    .FirstOrDefaultAsync(c => c.Id == currentSchoolId);

var currentRoom = currentSchool!
    .Rooms
    .FirstOrDefault(c => c.Number == number);
string firstName = $"Test-{Guid.NewGuid()}";
await _mediator.Send(new CreateTeacher.Command(currentSchoolId,
    currentRoom!.Id,
    firstName,
    "Test",
    21));
currentSchool = await _dbContext.Schools
    .FirstOrDefaultAsync(c => c.Id == currentSchoolId);

currentRoom = currentSchool!
    .Rooms
    .FirstOrDefault(c => c.Number == number);
var newTeacher = currentRoom!
    .Teachers
    .FirstOrDefault(c => c.FirstName == firstName);
Assert.NotNull(newTeacher);
Assert.Equal(firstName, newTeacher.FirstName);
```

```
}
```

Спочатку генерується унікальне ім'я для нової школи і викликається команда “CreateSchool”, щоб створити нову школу з цим ім'ям. Далі здійснюється пошук ідентифікатора створеної школи в базі даних.

Потім генерується унікальний номер для нової кімнати і викликається команда "CreateRoom”, щоб створити нову кімнату з цим номером. Знову здійснюється пошук поточної школи і кімнати в базі даних.

Далі генерується унікальне ім'я для нового вчителя і викликається команда “CreateTeacher”, щоб створити нового вчителя з цим ім'ям. Перевіряється, чи був вчитель успішно створений шляхом пошуку його в кімнаті за іменем.

Нарешті, за допомогою функції “Assert” перевіряється, що новий вчитель не є нульовим і його ім'я співпадає з очікуваним значенням.

Отже, цей код виконує створення нової школи, кімнати і вчителя в тестовому середовищі та перевіряє, чи були ці об'єкти успішно створені.

5. DeleteRoom

```
public static class DeleteRoom
{
public sealed record Command(Guid SchoolId, Guid RoomId) : IRequest<Unit>;
public sealed class Handler : IRequestHandler<Command, Unit>
{
private readonly TestDbContext _dbContext;
        public Handler(TestDbContext dbContext)
        {
            _dbContext = dbContext;
```

```
        }

        public async Task<Unit> Handle(Command request, CancellationToken cancellationToken)
{
            var currentSchool = await _dbContext.Schools
                .FirstOrDefaultAsync(c => c.Id == request.SchoolId);
            if (currentSchool is null)
            {
                throw new ArgumentNullException("School not found at database");
            }
            var currentRoom = currentSchool
                .Rooms
                .FirstOrDefault(c => c.Id == request.RoomId);
            if (currentRoom is null)
            {
                throw new ArgumentNullException("Room not found at database");
            }
            currentSchool.RemoveRoom(currentRoom);
            await _dbContext.SaveChangesAsync();
            return Unit.Value;
        }
}
}
```

Цей код представляє функціональність видалення кімнати зі школи.

У класі “DeleteRoom” міститься вкладений запит “Command”, який приймає

ідентифікатор школи “(SchoolId)” та ідентифікатор кімнати “(RoomId)”.

У вкладеному класі “Handler”, який реалізує інтерфейс “IRequestHandler<Command, Unit>”, виконується обробка запиту. Спочатку здійснюється пошук школи за заданим ідентифікатором в базі даних. Якщо школа не знайдена, викидається виняток “ArgumentNullException”.

Далі здійснюється пошук кімнати в школі за заданим ідентифікатором. Якщо кімната не знайдена, також викидається виняток “ArgumentNullException”.

Після цього викликається метод RemoveRoom школи для видалення кімнати.

Наостанок, зміни зберігаються в базі даних за допомогою “await _dbContext.SaveChangesAsync()”, і повертається значення “Unit.Value”.

Отже, цей код реалізує логіку видалення кімнати зі школи в тестовому середовищі.

5.1. DeleteRoom_ReturnsNull_WhenRoomDeletedSuccessfully

```
{
        string name = $"New school- {Guid.NewGuid()}";
        _ = await _mediator.Send(new CreateSchool.Command(name,
Guid.NewGuid().ToString(), "Test"));
        var currentSchoolId = await _dbContext.Schools
            .AsNoTracking()
            .Where(c => c.Name == name)
            .Select(c => c.Id)
            .FirstOrDefaultAsync();
        string number = Guid.NewGuid().ToString();
        _ = await _mediator.Send(new CreateRoom.Command(currentSchoolId,
            number,
            "Test"));
```

```
    var currentSchool = await _dbContext.Schools
        .AsNoTracking()
        .FirstOrDefaultAsync(c => c.Id == currentSchoolId);
    var currentRoom = currentSchool!
        .Rooms
        .FirstOrDefault(c => c.Number == number);
    await _mediator.Send(new DeleteRoom.Command(currentSchoolId,
        currentRoom!.Id));
    currentSchool = await _dbContext.Schools
        .AsNoTracking()
        .FirstOrDefaultAsync(c => c.Id == currentSchoolId);
    var exists = currentSchool!
        .Rooms
        .Any(c => c.Number == number);

    Assert.False(exists);
}
```

Спочатку створюється нова школа та кімната за допомогою викликів “CreateSchool.Command” та “CreateRoom.Command”. Згенеровані ідентифікатори школи та кімнати зберігаються в “currentSchoolId” та “number” відповідно.

Потім, за допомогою “await _mediator.Send(new DeleteRoom.Command(currentSchoolId, currentRoom!.Id))”, викликається команда на видалення кімнати зі школи за заданими ідентифікаторами.

Після виконання видалення, знову отримується інформація про школу з бази даних, але з параметром “.AsNoTracking()”, щоб отримати некешовані дані.

Наступною дією є перевірка, чи більше не існує кімнати з видаленим номером в списку

кімнат школи. Це виконується за допомогою перевірки “exists”, який буде “false,” якщо кімната була успішно видалена.

Завершуючим кроком є перевірка втілення тесту за допомогою “Assert.False(exists)”, що підтверджує, що кімната була успішно видалена зі школи.

Отже, цей код тестує функціональність видалення кімнати зі школи та перевіряє, чи була кімната дійсно видалена.

5.2. DeleteRoom_ThrowsException_WhenSchoolNotExists

```
{
    await Assert.ThrowsAsync<ArgumentNullException>(() => _mediator.Send(new DeleteRoom.Command(Guid.NewGuid(), Guid.NewGuid())));
}
```

Перший тест перевіряє, що якщо ми спробуємо видалити кімнату, але передамо недійсні ідентифікатори школи та кімнати, то виникне помилка “ArgumentNullException”. Тест перевіряє, чи виклик команди видалення кімнати з недійсними ідентифікаторами дійсно викликає помилку.

5.3. DeleteRoom_ThrowsException_WhenRoomNotExists

```
public async Task DeleteRoom_ThrowsException_WhenRoomNotExists()
    {
        string name = $"New school- {Guid.NewGuid()}";
        _ = await _mediator.Send(new CreateSchool.Command(name, Guid.NewGuid().ToString(), "Test"));
```

```
var currentSchoolId = await _dbContext.Schools
    .AsNoTracking()
    .Where(c => c.Name == name)
    .Select(c => c.Id)
    .FirstOrDefaultAsync();

await Assert.ThrowsAsync<ArgumentNullException>(() => _mediator.Send(new DeleteRoom.Command(currentSchoolId, Guid.NewGuid())));
}
```

Тест перевіряє, що якщо ми спробуємо видалити неіснуючу кімнату, але відправимо дійсний ідентифікатор школи і неіснуючий ідентифікатор кімнати, то також виникне помилка “ArgumentNullException”. Спочатку створюється нова школа, а потім шукається ідентифікатор цієї школи, викликається команда видалення кімнати з дійсним ідентифікатором школи та неіснуючим ідентифікатором кімнати. Тест перевіряє, чи виклик команди з неіснуючим ідентифікатором кімнати дійсно викликає помилку.

Таким чином, ці два тести перевіряють, що виникає помилка “ArgumentNullException” при спробі видалення кімнати з недійсними або неіснуючими ідентифікаторами школи та кімнати.

6. DeleteStudent

```
Public static class DeleteStudent
{
public sealed record Command(Guid StudentId, Guid SchoolId, Guid RoomId) :
```

```
IRequest<Unit>;

public sealed class Handler : IRequestHandler<Command, Unit>
{
private readonly TestDbContext _dbContext;
        public Handler(TestDbContext dbContext)
        {
            _dbContext = dbContext;
        }
        public async Task<Unit> Handle(Command request, CancellationToken
cancellationToken)
{
            var currentSchool = await _dbContext.Schools
            .FirstOrDefaultAsync(c => c.Id == request.SchoolId);
            if (currentSchool is null)
            {
                throw new ArgumentNullException("School not found at database");
            }
            var currentRoom = currentSchool
                .Rooms
                .FirstOrDefault(c => c.Id == request.RoomId);
            if (currentRoom is null)
            {
                throw new ArgumentNullException("Room not found at database");
            }
            var currentStudent = currentRoom.Students
                .FirstOrDefault(c => c.Id == request.StudentId);
```

```
        if (currentStudent is null)
        {
            throw new ArgumentNullException("Student not found at database");
        }
        currentRoom.RemoveStudent(currentStudent);
        await _dbContext.SaveChangesAsync();
        return Unit.Value;
    }
}
}
```

Клас “DeleteStudent” містить команду для видалення студента з кімнати школи. Команда “Command” приймає ідентифікатор студента “(StudentId)”, ідентифікатор школи “(SchoolId)” і ідентифікатор кімнати “(RoomId)” як параметри.

У внутрішньому класі “Handler” реалізовано обробник команди. У ньому здійснюється доступ до бази даних через контекст “TestDbContext”. У методі “Handle” обробляється команда “DeleteStudent”. Спочатку перевіряється, чи існує школа з заданим ідентифікатором. Якщо школа не знайдена, викликається виняток “ArgumentNullException”.

Далі перевіряється, чи існує кімната з заданим ідентифікатором у вказаній школі. Якщо кімната не знайдена, також викликається виняток “ArgumentNullException”.

Потім перевіряється, чи існує студент з заданим ідентифікатором у вказаній кімнаті. Якщо студент не знайдений, викликається виняток “ArgumentNullException”.

Якщо всі перевірки успішні, студент видаляється з кімнати за допомогою методу “RemoveStudent”, а зміни зберігаються в базі даних за допомогою “SaveChangesAsync”. На завершення, повертається значення “Unit.Value” для підтвердження успішного виконання команди.

6.1. DeleteStudent_ReturnsNull_WhenStudentDeletedSuccessfully

6.2. DeleteStudent_ThrowsException_WhenRoomNotExists

6.3. DeleteStudent_ThrowsException_WhenSchoolNotExists

6.4. DeleteStudent_ThrowsException_WhenStudentNotExists

```
{
    string name = $"New school- {Guid.NewGuid()}";
    _ = await _mediator.Send(new CreateSchool.Command(name,
Guid.NewGuid().ToString(), "Test"));
    var currentSchoolId = await _dbContext.Schools
      .AsNoTracking()
      .Where(c => c.Name == name)
      .Select(c => c.Id)
      .FirstOrDefaultAsync();
    string number = Guid.NewGuid().ToString();

    _ = await _mediator.Send(new CreateRoom.Command(currentSchoolId,
      number,
      "Test"));
    var currentSchool = await _dbContext.Schools
      .AsNoTracking()
      .FirstOrDefaultAsync(c => c.Id == currentSchoolId);
```

```
        var currentRoom = currentSchool!
            .Rooms
            .FirstOrDefault(c => c.Number == number);

        await Assert.ThrowsAsync<ArgumentNullException>(() => _mediator.Send(new
DeleteStudent.Command(Guid.NewGuid(), currentSchoolId, currentRoom!.Id)));
    }
```

Створюється нова школа з унікальним ім'ям.

Знаходиться ідентифікатор цієї школи за її іменем.

Генерується випадковий номер кабінету.

Створюється новий кабінет у цій школі зі згенерованим номером.

Знаходиться інформація про цю школу (без відстеження змін) за її ідентифікатором.

Знаходиться інформація про цей кабінет у знайденій школі за номером.

Виконується команда видалення студента, передаючи випадковий ідентифікатор студента, ідентифікатор школи та ідентифікатор кабінету.

Перевіряється, що генерується виключення ArgumentNullException, оскільки ідентифікатор студента не був правильно вказаний і студент не знайдений у базі даних.

7. DeleteTeacher

```
public static class DeleteTeacher
{
public sealed record Command(Guid TeacherId, Guid SchoolId, Guid RoomId) :
```

```
IRequest<Unit>;

public sealed class Handler : IRequestHandler<Command, Unit>
{
private readonly TestDbContext _dbContext;
        public Handler(TestDbContext dbContext)
        {
            _dbContext = dbContext;
        }
        public async Task<Unit> Handle(Command request, CancellationToken
cancellationToken)
{
            var currentSchool = await _dbContext.Schools
            .FirstOrDefaultAsync(c => c.Id == request.SchoolId);
            if (currentSchool is null)
            {
                throw new ArgumentNullException("School not found at database");
            }
            var currentRoom = currentSchool
                .Rooms
                .FirstOrDefault(c => c.Id == request.RoomId);
            if (currentRoom is null)
            {
                throw new ArgumentNullException("Room not found at database");
            }
            var currentTeacher = currentRoom.Teachers
                .FirstOrDefault(c => c.Id == request.TeacherId);
```

```
        if (currentTeacher is null)
        {
            throw new ArgumentNullException("Student not found at database");
        }
        currentRoom.RemoveTeacher(currentTeacher);
        await _dbContext.SaveChangesAsync();
        return Unit.Value;
    }
}
}
```

Команда “Command”: Це запис даних, який містить ідентифікатор вчителя “(TeacherId)”, ідентифікатор школи “(SchoolId)” та ідентифікатор кабінету “(RoomId)”, необхідні для видалення вчителя.

Обробник “Handler”: Це клас, який реалізує інтерфейс “IRequestHandler” для обробки команди видалення вчителя. У методі “Handle” отримуються ідентифікатори школи, кабінету та вчителя з бази даних і перевіряється їх наявність. Якщо школа або кабінет не знайдені, викидається виняток “ArgumentNullException”. Потім вчителя видаляють з кабінету за допомогою методу “RemoveTeacher”, а зміни зберігаються у базі даних за допомогою “SaveChangesAsync”. Після цього повертається значення “Unit”.

Отже, цей код виконує процес видалення вчителя з кабінету у школі, перевіряючи наявність школи, кабінету та вчителя у базі даних перед видаленням. Якщо якийсь з елементів не знайдено, виникає виняток.

7.1. DeleteTeacher_ReturnsNull_WhenTeacherDeletedSuccessfully

7.2. DeleteTeacher_ThrowsException_WhenRoomNotExists

7.3. DeleteTeacher_ThrowsException_WhenSchoolNotExists

7.4. DeleteTeacher_ThrowsException_WhenTeacherNotExists

3.2 Апробація результатів дослідження

При запуску системи тестується умовна “школа” , а саме створення школи, класів, вчителів та студентів. Виконується 16 тестів за короткий час , до 2х секунд (рис. 3.1)

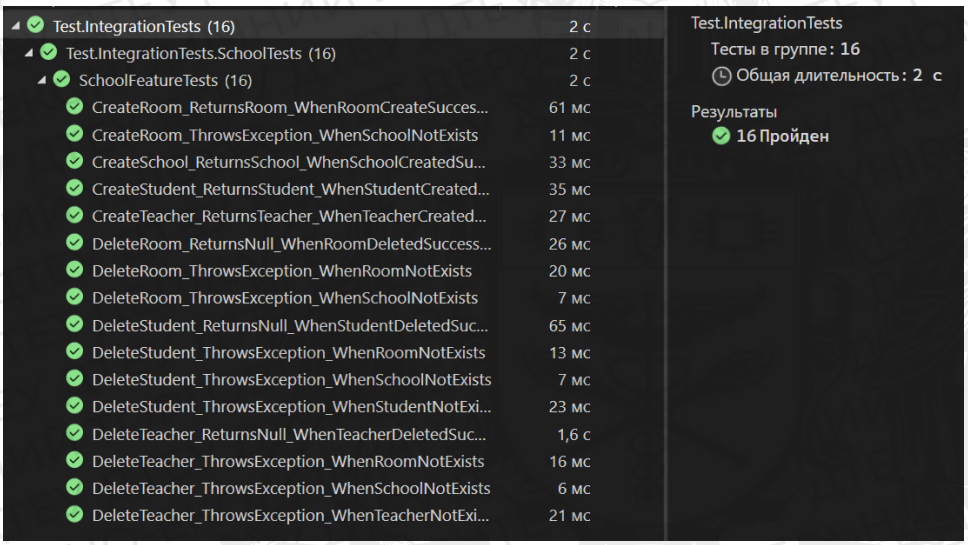


Рис 3.1

За допомогою команди “update-databese” оновлюємо базу даних для того, щоб вона відповідала останній версії програмного забезпечення. Коли змінюється структура або модель даних у додатку, виконується ця команда, щоб застосувати ці зміни до бази даних. Після того як данні завантажились переходим до MySql де бачимо що функції виконані успішно і данні дійсно відображаються(рис 3.2).

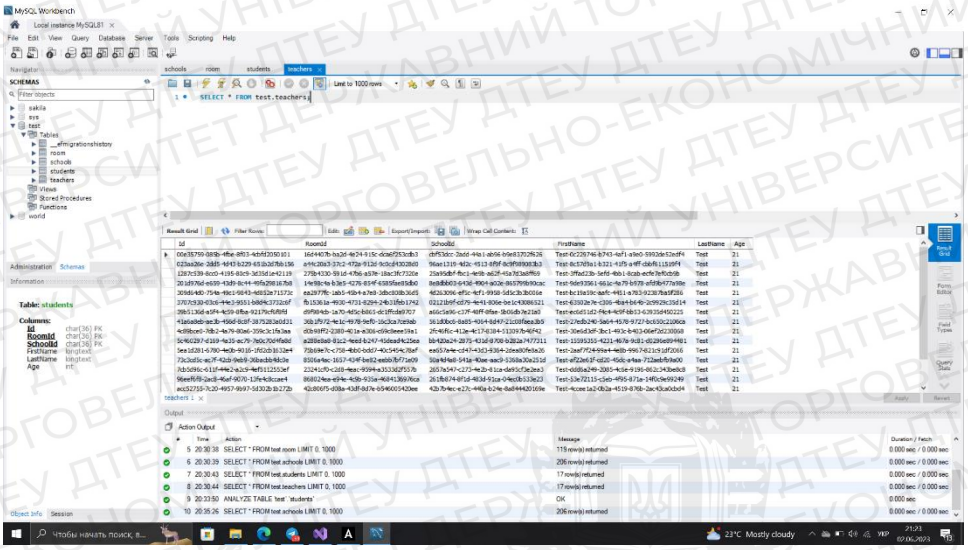
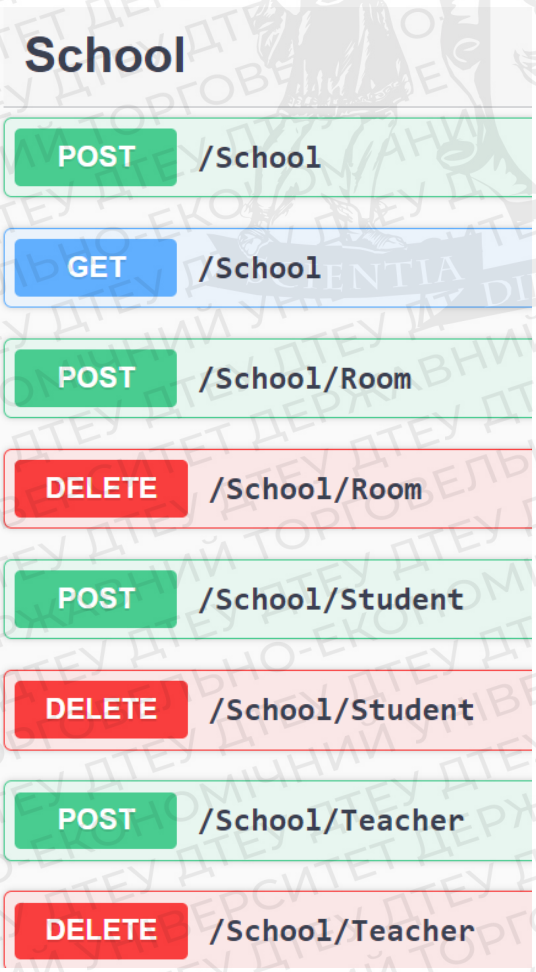
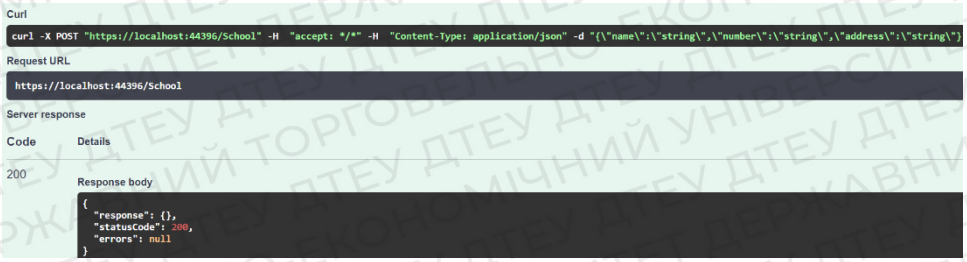


Рис 3.2

При старті системи нас перекидає на swagger - інструмент для тестування веб сервісів.



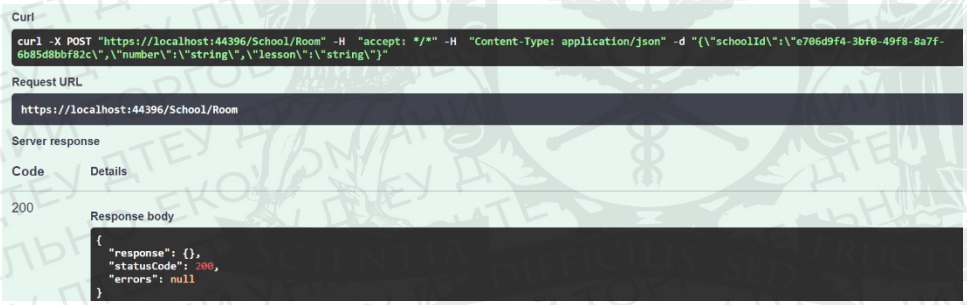
1. Post /School



curl - команда відправляє POST-запит з вказаними даними на вказану адресу. При успішному виконанні запиту, створена школа буде збережена на сервері.

Response body - показує, що запит був успішно оброблений сервером без помилок і не повертає додаткових даних у відповіді.

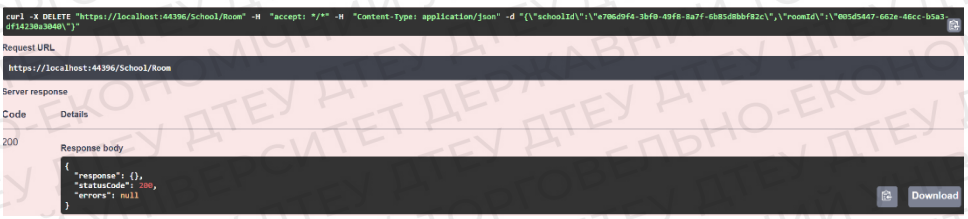
2. Post/School/Room



curl - відправляє POST-запит на вказаний URL з вказаними параметрами, передаючи дані у форматі JSON.

Response body - показує, що запит був успішно оброблений сервером без помилок і не повертає додаткових даних у відповіді.

3. Delete/school/room



Curl- відправляє DELETE-запит на вказаний URL з вказаними параметрами, передаючи дані у форматі JSON для видалення конкретної кімнати в рамках певної школи.

Response body - показує, що запит був успішно оброблений сервером без помилок і не повертає додаткових даних у відповіді.

Отже, на основі коду можна зробити висновок, що запит "CreateRoom, CreateSchool, CreateTeacher, CreateStudent, DeleteRoom, DeleteStudent, DeleteTeacher" виконує створення школи,кімнати в рамках школи, студента та вчителя в рамках кімнати та школи ,перевіряючи наявність школи і кімнати, і зберігаючи зміни в базі даних. Код виглядає достовірним і має потенціал для успішного виконання дій. Створення та видалення ресурсів "School, Room, Teacher та Student" в рамках API пройшло успішно без помилок.

ВИСНОВКИ

Система тестування веб-додатків є необхідним інструментом для забезпечення якості та надійності цих додатків перед їх релізом. Вона дозволяє виявляти та виправляти помилки та дефекти в ранніх стадіях розробки, що сприяє покращенню користувацького досвіду та забезпечує успішне функціонування додатків.

Для успішного створення системи тестування веб-додатків необхідно використовувати сучасні методики та підходи до тестування. Це включає в себе використання автоматизованих тестових інструментів, написання ефективних тестових сценаріїв та використання контрольних панелей для моніторингу процесу тестування.

Система тестування веб-додатків повинна бути гнучкою і масштабованою, з можливістю додавання нових тестів та модулів під час розвитку додатку. Вона також повинна враховувати змінні умови тестування, такі як різні конфігурації пристроїв та браузерів, що використовуються користувачами.

Розроблена і реалізована за допомогою сучасних програмних засобів система тестування яка умовно створює школу, класи, студентів та вчителів перевіряє чи успішно було все створенно та записано в базу данних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Andrew Lock Core in Action / Andrew Lock (832 pages, 2021)
2. E-Commerce: що це таке та як працює електронна комерція в Інтернеті [Електронний ресурс] - wezom.com.ua
3. Етапи та поради при тестуванні web-Додатків [Електронний ресурс] - https://qalight.ua/baza-znaniy/testuvannya-veb-proektiv-osnovni-etapi-ta-poradi
4. Glenford J. Myers, Мистецтво тестування програм (3-е видання)/ Гленфорд Майерс, Корі Сандлер, Том Баджетт. 272 с.
5. IEEE Standard for Software and System Test Documentation - IEEE 82-2008 [Електронний ресурс] - 829-2008 - IEEE Standard for Software and System Test Documentation | IEEE Standard | IEEE Xplore
6. IEEE Standard for Software Test Documentation [Електронний ресурс] - https://ieeexplore.ieee.org/document/741968
7. Integration Testing [Електронний ресурс] - https://www.softwaretestinghelp.com/integration-testing/
8. Марко Беллиньясо. Розробка Web-додатків в середовищі ASP.NET 2.0: завдання - проєкт - рішення = ASP.NET 2.0 Website Programming: Problem - Design – Solution/ Марко Беллиньясо (640 pages, 2007)
9. Кравець А.Ю. Автоматизація тестування web-сторінок за допомогою драйвера браузера Selenium WebDriver / І. В. Богач, А. Ю. Кравець
10. Односторінкові та багато сторінкові веб-сайти [Електронний ресурс] - "Single-page applications vs. multiple-page applications: pros, cons, pitfalls - BLAKIT - IT Solutions" (blak-it.com)
11. Особливості сучасного тестування Web-додатків [Електронний ресурс] - https://journals.khnu.km.ua/vestnik/wp-content/uploads/2022/07/vknu-ts-2022-n3-70-74.pdf

12. Roy Osherove. The Art of Unit Testing with Examples in .NET/ Roy Osherove (297 pages, 2009)
13. Тестування програмного забезпечення [Електронний ресурс] - https://er.chdtu.edu.ua/bitstream/ChSTU/1027/1/144.pdf
14. Тестування програмного забезпечення [Електронний ресурс] - https://www.youtube.com/watch?v=9Y6buO5QTHA
15. Тестування веб додатків за допомогою Apache Jmeter [Електронний ресурс] - JMeter Web Performance Testing (tutorialspoint.com)
16. Unit Testing [Електронний ресурс] - https://www.softwaretestinghelp.com/unit-testing/
17. Фреймворк Xunit [Електронний ресурс] -https://en.wikipedia.org/wiki/XUnit
18. Що таке інтеграційне тестування [Електронний ресурс] - https://www.softwaretestinghelp.com/what-is-integration-testing/
19. Як створити веб-додаток: типи, переваги, принцип роботи [Електронний ресурс] - https://wezom.com.ua/ua/blog/kak-sozdat-veb-prilozhenie
20. Як створювати високоефективні Web-додатки [Електронний ресурс] https://www.ranktracker.com/uk/blog/how-to-build-high-performing-web-apps-in-2022/