МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп'ютерних наук

ДІДУШОК Роман Зіновійович

Метод та програмне забезпечення для тестування backend частини веб-додатків / Method and Software for Testing the Backend Part of Web Applications

спеціальність: 121 - Інженерія програмного забезпечення
освітньо-професійна програма - Інженерія програмного забезпечення

Кваліфікаційна робота

Виконав студент групи ІПЗм-21
Р. З. Дідушок

_____

Науковий керівник:
к.т.н., доцент Р. П. Шевчук

_____

Кваліфікаційну роботу
допущено до захисту:

"___" _____ 20___ р.

Завідувач кафедри
_____ **А. В. Пукас**

ТЕРНОПІЛЬ - 2022

ЗМІСТ

ВСТУП

Актуальність теми. У процесі розробки програмного забезпечення його якість і надійність можуть змінюватися. Для оцінювання надійності ПЗ використовують різноманітні аналітичні методи, які можуть бути розділені на два основних класи: за областю визначення даних (статичні) та за часом (динамічні) або "моделі зростання надійності". Для забезпечення надійності програмних систем, починаючи з ранніх фаз створення, необхідно застосовувати тестування програмного забезпечення.

Тестування є невід'ємним етапом життєвого циклу програмного забезпечення. Зазвичай при тестуванні застосовуються класичні методи і техніки проектування тестів. Проте, якщо мова йде про веб-додатки, то існує ряд нюансів, пов'язаних із соціальними та технологічними особливостями веб-додатків, які відрізняють їх від інших видів додатків, і які обов'язково потрібно враховувати при тестуванні, щоб виконати його професійно.

Саме тому існує необхідність дотримання стратегії тестування, яка б покривала вимоги до тестування програмного коду веб-додатків, у яких важливим елементом є backend-частина. Стратегія тестування – це план проведення робіт з тестування системи або її модуля, що враховує специфіку функціональності і залежності з іншими компонентами системи і платформи.

Розробка методів та програмного забезпечення, які змогли б зменшити час на тестування являється однією з цілей розробки програмного засобу для тестування програмного коду backend частини веб-додатків.

Мета і задачі дослідження полягає в розробці програмного забезпечення для покращення процесів тестування програмного коду backend частини веб-сайтів, що дасть можливість користувачам, які не знають певної мови програмування, забезпечувати якість програмних продуктів за допомогою тестування API.

Для досягнення цієї мети необхідно вирішити наступні завдання:

- проаналізувати предметну область тестування програмного

забезпечення;

- дослідити вже створенні рішення та програмні засоби;

- модифікувати метод тестування програмного коду backend частини;

- спроектувати програмні засоби для тестування backend частини;

- розробити рішення, яке дозволить тестувати програмний код backend частини додатку через API-функції за допомогою інтерфейсу клієнта;

- провести тестування розробленого програмного забезпечення.

Об'єктом дослідження є тестування програмного коду backend частини веб-додатків.

Предмет дослідження: методи та алгоритми тестування програмного коду backend частини веб-додатків.

Методи дослідження: Для вирішення поставлених у кваліфікаційній роботі завдань використано такі методи дослідження: теорію ймовірності та математичну статистику; теорію системного аналізу та теорія математичного моделювання; теорію алгоритмів та принципи об’єктно-орієнтованого програмування.

Наукова новизна отриманих результатів.

Удосконалено метод тестування програмного коду backend частини веб-додатків, який на відміну від наявних, забезпечує тестера візуальним інтерфейсом, що дає змогу візуально оцінити важливість тестування кожної програмної функції, і в такий спосіб підвищити ефективність процесу тестування.

Практичне значення одержаних результатів. Розроблено програмне для тестування програмного коду backend частини веб-додатків, яке дає змогу виконувати тестування виконання програмного коду без програмування у візуальному режимі.

Особистий внесок магістранта

Всі результати отримані автором самостійно.

Апробація результатів

Основні положення магістерського дослідження апробовані на ІІІ Міжнародній науково-практичній онлайн-конференції «Актуальні проблеми, пріоритетні напрямки та стратегії розвитку України» 13 жовтня у м. Київ.

Публікації.

1. Шевчук Р.П. Особливості потокової передачі мультимедійного трафіку / Р.П. Шевчук, М.В. Клибаник, В.В. Кремінський // Актуальні проблеми, пріоритетні напрямки та стратегії розвитку України: тези доповідей ІІІ Міжнародної науково-практичної онлайн-конференції, м. Київ, 13 жовтня 2021 року/ редкол. О.С. Волошкіна та ін. – К.: ІТТА, 2021. – с. 1363- 1365

РОЗДІЛ І
АНАЛІЗ МАТЕМАТИЧНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ТЕСТУВАННЯ ВЕБ-ДОДАТКІВ

1.1. Аналіз стану предметної області

Тестування програмного забезпечення – це метод перевірки, чи відповідає фактичний програмний продукт очікуваним вимогам, і переконатися, що програмний продукт не містить дефектів. Він включає виконання програмних/системних компонентів за допомогою ручних або автоматизованих інструментів для оцінки однієї або кількох властивостей, що цікавлять. Метою тестування програмного забезпечення є виявлення помилок, прогалин або відсутніх вимог на відміну від реальних вимог.

Тестування програмного забезпечення важливе, оскільки якщо в програмному забезпеченні є помилки або дефекти, їх можна виявити завчасно і вирішити до доставки програмного продукту. Правильно перевірений програмний продукт забезпечує надійність, безпеку та високу продуктивність, що в подальшому призводить до економії часу, економічної ефективності та задоволення клієнтів.

Тестування важливе, оскільки помилки програмного забезпечення можуть бути дорогими або навіть небезпечними. Помилки програмного забезпечення потенційно можуть призвести до грошових і людських втрат, і історія повна таких прикладів.

У квітні 2015 року термінал Bloomberg в Лондоні зазнав збою через програмний збій, який вплинув на понад 300 000 трейдерів на фінансових ринках. Це змусило уряд відкласти продаж боргу на суму 3 мільярди фунтів стерлінгів.

Автомобілі Nissan відкликали з ринку понад 1 мільйон автомобілів через збій програмного забезпечення в сенсорних датчиках подушок безпеки. Повідомлялося про дві аварії через збій цього програмного забезпечення.

Starbucks був змушений закрити близько 60 відсотків магазинів у США та Канаді через збій програмного забезпечення в її POS-системі. У якийсь момент у магазині безкоштовно подавали каву, оскільки вони не змогли обробити транзакцію.

Деякі сторонні роздрібні продавці Amazon помітили, що їхня ціна продукту знижена до 1 пенса через програмний збій. Вони зазнали великих втрат.

Вразливість у Windows 10. Ця помилка дозволяє користувачам вийти з пісочниці безпеки через недолік у системі win32k.

У 2015 році винищувач F-35 став жертвою програмної помилки, через яку він не міг правильно виявляти цілі.

Airbus A300 China Airlines розбився через програмну помилку 26 квітня 1994 року, в результаті чого загинули 264 невинні людини.

У 1985 році канадський апарат променевої терапії Therac-25 вийшов з ладу через програмну помилку і доставив смертельні дози опромінення пацієнтам, в результаті чого 3 людини загинули і 3 отримали тяжкі поранення.

У квітні 1999 року помилка програмного забезпечення спричинила невдачу запуску військового супутника вартістю 1,2 мільярда доларів, що стало найдорожчим нещасним випадком в історії.

Економічна ефективність: це одна з важливих переваг тестування програмного забезпечення. Своєчасне тестування будь-якого IT-проекту допоможе заощадити гроші в довгостроковій перспективі. Якщо помилки виявлені на ранньому етапі тестування програмного забезпечення, їх виправлення коштує дешевше.

Безпека: це найбільш вразлива і чутлива перевага тестування програмного забезпечення. Люди шукають надійні продукти. Це допомагає усунути ризики та проблеми раніше.

Якість продукту: це важлива вимога будь-якого програмного продукту.

Тестування гарантує, що якісний продукт буде доставлений клієнтам.

Задоволеність клієнта: головна мета будь-якого продукту - задовольнити своїх

клієнтів. Тестування UI/UX забезпечує найкращий досвід користувача.

Відповідно до ANSI/IEEE 1059, тестування в інженерії програмного забезпечення — це процес оцінки програмного продукту, щоб визначити, чи відповідає поточний програмний продукт необхідним умовам чи ні. Процес тестування включає оцінку функцій програмного продукту на вимоги з точки зору будь-яких відсутніх вимог, помилок або помилок, безпеки, надійності та продуктивності.

Зазвичай тестування поділяють на три категорії:

- функціональне тестування,
- нефункціональне тестування або тестування продуктивності,
- технічне обслуговування (регресія та обслуговування) [2].

Таблиця 1.1 - Категорії та типи тестування

Категорія тестування	Типи тестування
Функціональне тестування	Модульне тестування Інтеграційне тестування Димне тестування UAT (прийомне тестування користувача) Локалізація Глобалізація Сумісність
Нефункціональне тестування	Продуктивність Витривалість Навантажувальне Обсяг Масштабованість Зручність використання
Технічне обслуговування	Регресія Технічне обслуговування

Важливі стратегії в розробці програмного забезпечення:

1. Модульне тестування: цього базового підходу до тестування програмного забезпечення програміст дотримується для перевірки блоку програми. Це допомагає розробникам знати, чи працює окремий блок коду належним чином.

2. Інтеграційне тестування: зосереджено на побудові та дизайні

програмного забезпечення. Перевірка, що інтегровані блоки працюють без помилок чи ні.

3. Тестування системи: у цьому методі програмне забезпечення компілюється як єдине ціле, а потім тестується як єдине ціле. Ця стратегія тестування перевіряє функціональність, безпеку, портативність тощо.

4. Тестування програм у тестуванні програмного забезпечення – це метод виконання фактичної програми з метою перевірки поведінки програми та пошуку помилок. Програма виконується з даними тестового прикладу для аналізу поведінки програми або реакції на дані тесту. Хороше тестування програми - це те, що має високі шанси знайти помилки.

Тестування програмного забезпечення визначається як діяльність, спрямована на перевірку відповідності фактичних результатів очікуваним результатам і забезпечення відсутності дефектів у системі програмного забезпечення.

Тестування важливе, оскільки помилки програмного забезпечення можуть бути дорогими або навіть небезпечними.

Важливими причинами використання тестування програмного забезпечення є: рентабельність, безпека, якість продукції та задоволення клієнтів.

Зазвичай тестування поділяється на три категорії: функціональне тестування, нефункціональне тестування або тестування продуктивності та обслуговування.

Важливими стратегіями в розробці програмного забезпечення є: модульне тестування, інтеграційне тестування, перевірочне тестування та тестування системи.

1.2. Порівняльний аналіз аналогів

На сьогоднішній день існує багато різних програмних додатків для тестування API з своїми недоліками і перевагами. Нижче наведено порівняльний аналіз аналогів до додатку, що розроблявся у магістерській кваліфікаційній роботі.

Платформа Postman (рис. 1.1) - призначена для перевірки запитів з клієнта на сервер та отримання відповіді від бекенда [3]. Postman є ПЗ для API тестування для тих, хто не бажає мати справи з кодуванням в інтегрованому середовищі розробки, використовуючи ту ж мову програмування, що й розробник [3]. Недоліком цього ПЗ є відсутність інтерфейсу українською мовою та складний інтерфейс.

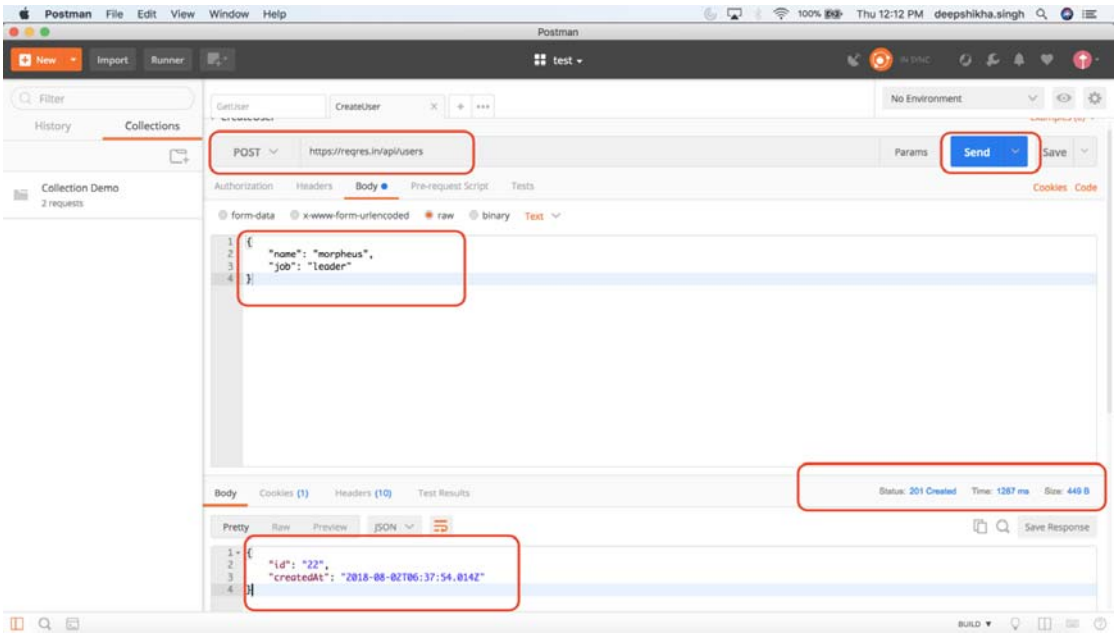


Рис. 1.1. Інтерфейс платформи Postman

SoapUI - консольний інструмент, призначеним для тестування API і дозволяє користувачам тестувати API REST і SOAP, а також Web-сервіси (рис. 1.2). За допомогою SoapUI користувачі можуть отримати повний вихідний документ і вбудувати набір функцій [3]. Створити тест легко та швидко за допомогою технологій перетягування об'єктів «мишкою» (Drag and drop) та методу «вказівки та клацання» (Point-and-click), швидко створити код

користувача за допомогою Groovy.

Потужне тестування на основі даних: Дані завантажуються з файлів, баз даних та Excel, тому вони можуть створити симуляцію взаємодії користувача та API.

Створення комплексних сценаріїв та підтримка асинхронного тестування. Повторне використання скриптів: завантаження тестів і сканування безпеки можуть повторно використовуватися у разі функціонального тестування лише за кілька кроків. Недоліком даного ПЗ є відсутність інтерфейсу українською мовою, платна основа користування та складний і незрозумілий інтерфейс.

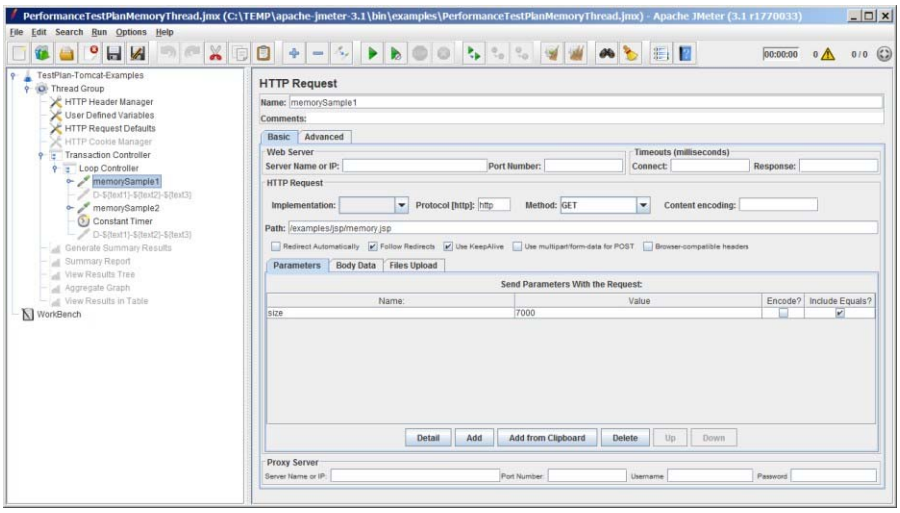


Рис.1.2. Інтерфейс SoapUI

JMeter - широко використовується для функціонального тестування API, проте спочатку він створювався для тестування навантаження (рис.1.3) [4]. Підтримує відтворення результатів тестування. Автоматично працює з файлами CSV, що дозволяє команді швидко створювати унікальні значення параметрів для тестування API. Завдяки інтеграції між JMeter та Jenkins, користувачі можуть включати тести API у конвеєрні обробки CI.

Цей інструмент може використовуватися як для статичного, так і для динамічного тестування продуктивності ресурсів. Недоліком даного ПЗ є відсутність інтерфейсу українською мовою, складна процедура налаштування середовища для тестування, більше підходить для навантажувального тестування.

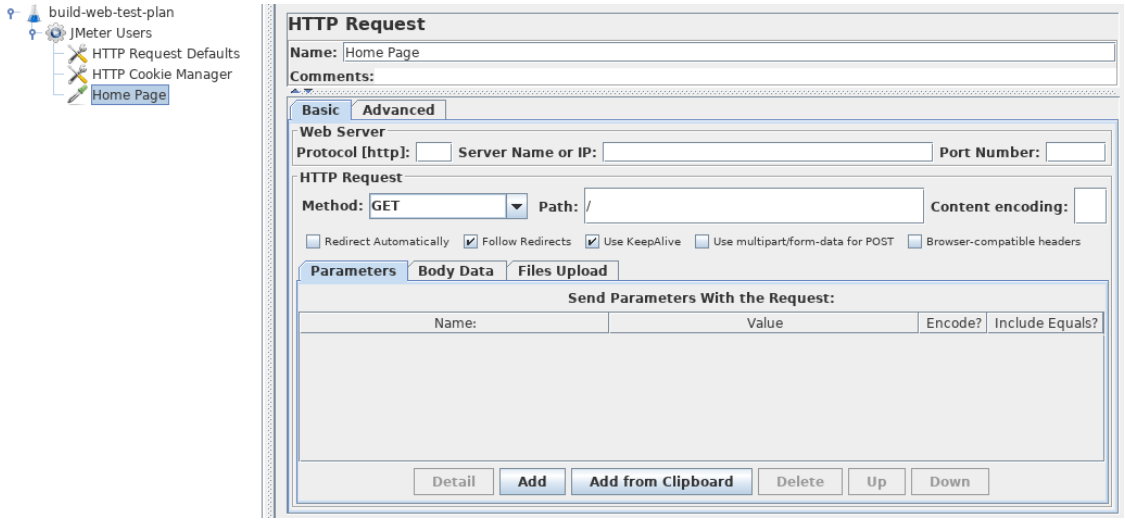


Рис.1.3. Інтерфейс JMeter

Враховуючи недоліки зазначених вище програм можна зробити висновок,що розробка власної реалізації є доцільною. Буде розроблено веб-додаток українською мовою для тестування API.

1.3. Особливості роботи веб-додатків

HTTP (Hypertext Transfer Protocol) визначає безліч методів запитів, які вказують, яка бажана дія виконається для певного ресурсу. Незважаючи на те, що їхні назви можуть бути іменниками, ці методи запиту іноді називаються дієсловами HTTP.

CRUD - (англ. create read update delete - "Створення читання оновлення видалення") скорочене найменування 4 базових функцій при роботі з персистентними сховищами даних - створення, читання, редагування та видалення.

Операції у HTTP

- Створення (Create) POST
- Читання (Read) GET
- Редагування (Update) PUT або PATCH
- Видалення (Delete) DELETE

Ідемпотентність - це властивість об'єкта чи операції при повторному застосуванні операції до об'єкта давати той самий результат, що й за першому.

Ідемпотентні методи – GET, HEAD, OPTIONS, TRACE, + (Небезпечні PUT або DELETE).

Кожен реалізує свою семантику, але кожна група команд поділяє загальні властивості: так, методи можуть бути безпечними (не змінюють стану сервера – GET, HEAD, OPTIONS), ідемпотентними (повертають той самий результат на ідентичний запит – GET, HEAD, PUT, DELETE) або кешуються.

Методи HTTP запиту:

- GET – запитує подання ресурсу. Запити з цього методу можуть лише витягувати дані. Немає тіла.
- POST - використовується для надсилання сутностей до певного ресурсу.
- Часто викликає зміну стану або якісь побічні ефекти на сервері. Має тіло.
- PUT – замінює всі поточні уявлення ресурсу даними запиту. DELETE – видаляє зазначений ресурс.
- PATCH – використовується для часткової зміни ресурсу.

POST запит застосовується передачі даних заданому ресурсу. Наприклад, у блогах відвідувачі зазвичай можуть вводити свої коментарі до записів у HTML-форму, після чого вони передаються серверу методом POST і він поміщає їх на сторінку. При цьому дані (у прикладі з блогами — текст коментаря) включаються в тіло запиту. Аналогічно, за допомогою методу POST зазвичай завантажуються файли на сервер. Повідомлення відповіді сервера виконання методу POST не кешується.

Відмінність GET від POST полягає в тому, що GET надсилає запит на отримання даних, POST надсилає дані. GET. Додається до закладок. Кешується.

Історія залишається у закладках браузера. Є обмеження за символами, так як дані передаються в URL, то повинен обмежуватися в 2048 символах (мах рядок символів в URL). За типом даних можна використовувати лише символи ASCII. Менш безпечний, оскільки дані, що передаються в URL, видно користувачу. Дані URL доступні всім.

Різниця між PUT і POST полягає в тому, що PUT є ідемпотентним: повторне його застосування дає той же результат, що і при першому

застосуванні (тобто метод не має побічних ефектів), тоді як повторний виклик одного і того ж методу POST може мати такі ефекти, як, наприклад, оформлення одного і того ж замовлення кілька разів.

Всі безпечні методи також є ідемпотентними, як і деякі інші, але при цьому небезпечні, такі як PUT або DELETE.

Структура HTTP запиту:

- рядок запиту, в якій вказується версія HTTP протоколу і HTTP метод запиту;
- нуль або кілька заголовків, розділених між собою символом кінця рядка, в яких передаються інші HTTP праметри для успішного з'єднання HTTP;
- порожній рядок, щоб відокремити службову інформацію від тіла повідомлення;
- необов'язкове тіло повідомлення.

Header потрібен для того, щоб комп'ютер міг розуміти з яким ресурсом працювати:

Заголовок-сутність Content-Type використовується у тому, щоб визначити MIME тип ресурсу.

MIME тип: Content-Type (text/html; charset=utf-8)

Клієнт може встановити Accept в application/json, якщо він запитує відповідь JSON.

І навпаки, коли дані надсилаються, встановлений Content-Type в application/xml говорить клієнту, що дані були відправлені в XML формі.

Приклад запит-відповідь за протоколом HTTP [5] наведено на рисунку 1.4.

Рис. 1.4 - Приклад запиту і відповіді за протоколом HTTP

Отже, після проведеного аналізу методів для реалізації тестування системи, було вирішено використовувати CRUD методи, так як самі ці методи дозволяють отримувати дані з сервера, записувати данні та здійснювати інші операції з даними для тестування backend за допомогою API.

1.4. Постановка задачі дослідження

Після аналізу стану програмних засобів для тестування, порівняння існуючих аналогів та аналізу методів і засобів реалізації програмного продукту було визначено наступні завдання, які необхідно виконати для розробки програмного продукту:

- розробити програмний додаток для тестування API;
- розробити модуль отримання даних;
- розробити модуль відправки даних;
- розробити серверну частину;
- розробити клієнтську частину програмного продукту;
- провести тестування програмного продукту.

Висновки до розділу І

В першому розділі проаналізовано програмні засоби для тестування веб-додатків. Було розглянуто такі аналоги як: Postman, SoapUI, JMeter. Порівняння уже створених аналогів дозволило встановити, що є необхідність розробити власний додаток. Також було проведено аналіз методів реалізації програмного продукту. Було обрано методи роботи з даними. Було розглянуто методи Get, Post, Put, Delete,Patch. Після порівняння було обрано методи Get, Post, Patch, Delete. В результаті було розроблено основні завдання, які необхідно виконати при розробці програмного забезпечення.

РОЗДІЛ ІІ
МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ТЕСТУВАННЯ BACKEND ЧАСТИНИ ВЕБ-ДОДАТКІВ

2.2. Методи і принципи тестування веб-додатків

Тестування ПЗ - складний процес з великими затратами ресурсів та великою різновидністю, типами і структурою тестування. При цьому використовуються різні методи тестування.

Нижче наведено різні типи тестування, які можуть бути використані для тестування програмного забезпечення.

Ручне тестування - включає тестування програмного забезпечення вручну, тобто без використання будь-якого автоматизованого інструменту або будь-якого сценарію. У цьому типі тестер бере на себе роль кінцевого користувача та тестує програмне забезпечення для виявлення будь-яких непередбачених дій чи помилок. Існують різні етапи ручного тестування, такі як одиничне тестування, інтеграційне тестування, тестування системи та приймальні випробування користувачів.

Тестери використовують плани тестування, тестові приклади або тестові сценарії для тестування програмного забезпечення, щоб забезпечити повноту тестування. Ручне тестування також включає розвідувальне тестування, оскільки тестери досліджують програмне забезпечення для виявлення помилок в ньому.

Тестування автоматизації, яке також відоме як Test Automation – це коли тестер пише сценарії та використовує інше програмне забезпечення для тестування продукту. Цей процес включає автоматизацію ручного процесу. Тестування автоматизації використовується для повторного запуску тестових сценаріїв, які виконувались вручну, швидко та багаторазово.

Крім регресійного тестування, тестування автоматизації також використовується для тестування програми з точки зору навантаження,

продуктивності та напруги. Це збільшує охоплення тестування, підвищує точність та заощаджує час та гроші в порівнянні з ручним тестуванням.

Автоматизувати все у програмному забезпеченні неможливо. Області, в яких користувач може здійснювати транзакції, такі як форма входу або форми реєстрації, в будь-якій області, де багато користувачів можуть отримати доступ до програмного забезпечення одночасно, повинні бути автоматизовані.

Крім того, всі елементи GUI, з'єднання з базами даних, перевірки полів тощо, можуть бути ефективно протестовані шляхом автоматизації ручного процесу.

Автоматизація тестування повинна використовуватися під час розгляду наступних аспектів програмного забезпечення:

- Великі та критичні проекти
- Проекти, що потребують частого тестування тих самих областей
- Вимоги не часто змінюються
- Доступ до програми для завантаження та продуктивності з багатьма віртуальними користувачами
- Стабільне програмне забезпечення щодо ручного тестування
- Доступність часу

Автоматизація виконується за допомогою підтримуючої комп'ютерної мови, такої як скрипти VB та автоматизованої програмної програми. Існує безліч доступних інструментів, які можна використовувати для написання сценаріїв автоматизації [6].

Процес, який можна використовувати для автоматизації процесу тестування:

- Визначення областей у програмному забезпеченні для автоматизації
- Вибір відповідного інструменту для автоматизації тестування
- Написання тестових скриптів
- Розробка тестових костюмів
- Виконання скриптів
- Створення звітів про результати

- Тестування ПЗ - Методи

Існують різні способи, які можна використовувати для тестування програмного забезпечення, наприклад Тестування Black-Box.

Методика тестування без будь-яких знань про внутрішню роботу програми називається «чорною скринькою». Тестер не звертає уваги на архітектуру системи та не має доступу до вихідного коду. Як правило, при виконанні тесту з «чорною скринькою» тестер буде взаємодіяти з інтерфейсом системи, надаючи вхідні дані та аналізуючи виходи, не знаючи, як і де обробляються входи.

У таблиці 2.1 перераховані переваги та недоліки тестування чорної скриньки.

Таблиця 2.1 - Переваги і недоліки тестування чорної скриньки

Переваги	Недоліки
Добре підходить і ефективний для великих сегментів коду.	Обмежене покриття, оскільки насправді виконується лише обрана кількість тестових сценаріїв.
Кодовий доступ не потрібний.	Неефективне тестування через те, що тестер тільки має обмежені знання про додаток.
Чіткий поділ перспективи користувача з погляду розробника з допомогою певних ролей.	Сліпе охоплення, оскільки тестер не може орієнтуватися на певні сегменти коду або області помилок.
Велика кількість помірковано кваліфікованих тестувальників може протестувати програму без будь-яких знань про реалізацію, мову програмування або операційні системи.	Тестові приклади важко розробити.

Також можливе тестування білої скриньки. Тестування білої скриньки – це докладне дослідження внутрішньої логіки та структури коду. Тестування з

використанням білої скриньки також називається тестуванням скла абовідкритим тестуванням. Щоб виконати тестування білої скриньки у програмі,тестер повинен знати внутрішню роботу коду.

Тестер повинен заглянути всередину вихідного коду та з'ясувати, який пристрій/блок коду поводиться некоректно.

У таблиці 2.2 перераховані переваги та недоліки тестування білої скриньки.

Таблиця 2.2 - Переваги і недоліки тестування білої скриньки

Переваги	Недоліки
Оскільки тестер знає вихідний код, дуже легко дізнатися, який тип даних може допомогти в ефективному тестуванні програми.	У зв'язку з тим, що для тестування білихящиків потрібен кваліфікований тестер, витрати збільшуються.
Це допомагає в оптимізації коду.	Іноді неможливо заглянути в кожен куточок і кут, щоб виявити прихованіпомилки, які можуть створювати проблеми, оскільки багато шляхівбудуть неперевірені.
Додаткові рядки коду можуть бути видалені, що може призвести до прихованих дефектів.	Важко підтримувати тестування білихящиків, оскільки для цього потрібні спеціалізовані інструменти, такі як аналізатори коду та налагодження.
Завдяки знанням тестера про код, максимальне охоплення досягається при написанні сценарію сценарію.	

Тестування сірої скриньки – це метод тестування програми з обмеженим знанням внутрішньої роботи програми.

Освоєння домену системи завжди дає тестеру перевагу над кимось із обмеженими знаннями домену. На відміну від тестування чорної скриньки, де тестер тестує тільки інтерфейс користувача програми; при тестуванні в сірому польоті тестер має доступ до проектної документації та бази даних. Маючи ці

знання, тестер може підготувати найкращі тестові дані та сценарії тестування під час складання плану тестування.

Таблиця 2.3 - Переваги і недоліки тестування сірої скриньки.

Переваги	Недоліки
Пропонує комбіновані переваги тестування чорної скриньки та білої скриньки, де це можливо.	Оскільки доступ до вихідного коду недоступний, можливість пройти через код та зону тестування обмежена.
Тестувальники сірої скриньки не покладаються на вихідний код; натомість вони покладаються визначення інтерфейсу і функціональні специфікації.	Тести можуть бути надмірними, якщо розробник програмного забезпечення вже виконав тестовий приклад.
Грунтуючись на наявній обмеженій інформації, тестер сірої скриньки може розробити відмінні сценарії тестування, особливо щодо протоколів зв'язку та обробки даних.	Тестування всіх можливих вхідних потоків нереально, оскільки цього потрібно необгрунтоване кількість часу; тому багато програмні шляхи будуть неперевірені.
Тест виконується з погляду користувача, а не дизайнера.	

У таблиці 2.4 перераховані точки, які розрізняють тестування «чорної скриньки», «сірої скриньки» та «білої скриньки».

Таблиця 2.4 - Порівняння методик тестування

Тестування Black-Box	Тестування сірих скриньок	Тестування білої скриньки
Не потрібно знати внутрішню роботу програми.	Тестер має обмежене знання внутрішньої роботи програми.	Тестер має повне уявлення про внутрішню роботу програми.
Також відомий як тестування із закритою скринькою, тестування з використанням даних або функціональне тестування.	Також відомий як прозоре тестування, оскільки тестер має обмежене знання внутрішніх аспектів програми.	Також відомий як прозоре тестування, структурне тестування чи тестування на основі коду.

Виконується кінцевими користувачами, а також тестувальниками та розробниками.	Виконується кінцевими користувачами, а також тестувальниками та розробниками.	Зазвичай виконуються тестувальниками та розробниками.
Тестування ґрунтується на зовнішніх очікуваннях. Внутрішня поведінка програми невідома.	Тестування виконується на основі діаграм бази даних високого рівня та діаграм потоків даних.	Внутрішні роботи повністю відомі і тестер може відповідним чином створювати тестові дані.
Він є вичерпним та найменш трудомістким.	Частково трудомісткий та вичерпний.	Найбільш вичерпний та трудомісткий тип тестування.
Чи не підходить для тестування алгоритмів.	Чи не підходить для тестування алгоритмів.	Підходить для тестування алгоритмів.
Це можна зробити лише методом спроб та помилок.	Домени даних та внутрішні межі можуть бути перевірені, якщо вони відомі.	Домени даних та внутрішні кордони можуть бути краще перевірені.

Під час тестування є різні рівні тестування. Рівні тестування включають різні методології, які можна використовувати під час проведення тестування програмного забезпечення [7]. Основними рівнями тестування програмного забезпечення є:

- Функціональне тестування
- Нефункціональне тестування

Функціональне тестування - це тип тестування «чорної скриньки», що базується на специфікаціях програмного забезпечення, яке має бути перевірено. Програма перевіряється шляхом введення, а потім перевіряється результат, який повинен відповідати функціям, для яких він призначався. Функціональне тестування програмного забезпечення проводиться у повній інтегрованій системі з метою оцінки відповідності системи зазначеним вимогам.

Під час тестування програми для роботи є п'ять кроків.

Таблиця 2.5 - Кроки функціонального тестування

Крок	Опис
I	Визначення функціональності, призначеної для передбачуваної програми
II	Створення тестових даних на основі специфікацій програми
III	Результат заснований на тестових даних та специфікаціях програми.
IV	Написання тестових сценаріїв та виконання тестових прикладів.
V	Порівняння фактичних та очікуваних результатів на основі виконаних тестових випадків

Ефективна практика тестування включає наведені вище кроки, що застосовуються до політик тестування кожної організації, і, отже, вона стежитиме за тим, щоб організація дотримувалася найсуворіших стандартів, коли йдеться про якість програмного забезпечення.

Тестування пристрою - цей тип тестування виконується розробниками перед тим, як установка буде передана групі тестування для офіційного виконання тестових прикладів. Модульне тестування виконується відповідними розробниками в окремих одиницях, призначених вихідним кодом областях. Розробники використовують тестові дані, відмінні від тестових даних команди забезпечення якості.

Мета модульного тестування полягає в тому, щоб ізолювати кожну частину програми та показати, що окремі частини є правильними з погляду вимог та функціональності.

Тестування не може зловити будь-яку помилку у додатку. Неможливо оцінити кожен шлях виконання у кожному програмному додатку. Те саме відбувається з модульним тестуванням.

Існує обмеження на кількість сценаріїв та тестових даних, які розробник може використовувати для перевірки вихідного коду. Після вичерпання всіх опцій немає вибору, крім припинити модульне тестування і об'єднати сегмент коду з іншими модулями.

Тестування інтеграції визначається як тестування комбінованих частин програми, щоб визначити, чи правильно вони функціонують. Інтеграційне

тестування може бути виконане двома способами: інтеграційне тестування знизу вгору та тестування інтеграції зверху вниз.

Метод тестування інтеграції знизу догори - це тестування починається з модульного тестування, за яким слідують тести прогресивно більш високих комбінацій модулів, які називаються модулями або складаннями.

Інтеграція зверху вниз - у цьому тестуванні модулі найвищого рівня тестуються спочатку та поступово, потім тестуються модулі нижнього рівня.

У комплексному середовищі розробки програмного забезпечення спочатку виконується тестування знизу нагору, а потім тестування зверху вниз. Процес завершується декількома випробуваннями повної програми, переважно в сценаріях, призначених для імітації реальних ситуацій.

Тестування системи тестує систему загалом. Після того, як всі компоненти інтегровані, програма в цілому перевіряється суворо, щоб переконатися, що вона відповідає зазначеним стандартам якості. Цей тип тестування виконується спеціалізованою групою тестування.

Системне тестування важливе з таких причин:

- Тестування системи є першим кроком у життєвому циклі розробки програмного забезпечення, де програма протестована в цілому.
- Додаток перевірено ретельно, щоб переконатися, що він відповідає функціональним та технічним вимогам.
- Додаток тестується в середовищі, яке дуже близьке до робочого середовища, в якому програма буде розгорнута.
- Системне тестування дозволяє нам тестувати, перевіряти та перевіряти як бізнес-вимоги, так і архітектуру програми.
- Регресійне тестування

Щоразу, коли робиться зміна в програмному додатку, цілком можливо, що ця зміна вплинула інші області програми. Регресійне тестування виконується для перевірки того, що виправлена помилка не призвела до іншої функціональності чи порушення бізнес-правил. Мета регресійного тестування

полягає в тому, щоб гарантувати, що зміна, така як виправлення помилки, не повинна призводити до виявлення іншої проблеми в додатку.

Регресійне тестування важливе з таких причин:

- Мінімізує пробіли під час тестування, коли необхідно перевірити програму із внесеними змінами.
- Перевіряє нові зміни, щоб переконатися, що внесені зміни не вплинули на будь-яку іншу область програми.
- Пом'якшує ризики, коли регресійне тестування виконується у додатку.
- Тестове охоплення збільшується без шкоди термінів.
- Збільшує швидкість виходу ринку продукту.
- Приймальне тестування

Це, можливо, найважливіший тип тестування, оскільки він проводиться командою забезпечення якості, яка визначатиме, чи відповідає додаток необхідним специфікаціям і задовольняє вимогам клієнта. Команда QA буде мати набір попередньо написаних сценаріїв та тестових прикладів, які будуть використовуватись для тестування програми.

Буде представлено більше ідей про додаток, і на ньому можна буде провести більше тестів, щоб оцінити його точність та причини, з яких був ініційований проект. Приймальні тести призначені не лише для вказівки простих орфографічних помилок, косметологічних помилок або пробілів в інтерфейсі, але й для вказівки на наявність помилок у додатку, які можуть призвести до збоїв системи або серйозних помилок у додатку.

Виконуючи приймальні випробування у додатку, група тестування зменшить, як додаток працюватиме з виробництва. Існують також юридичні та контрактні вимоги для прийняття системи.

Альфа-тестування - цей тест є першим етапом тестування і виконуватиметься серед команд (розробників та команд QA). Тестування модулів, тестування інтеграції та тестування системи у поєднанні один з одним відомі як альфа-тестування. На цьому етапі у додатку будуть протестовані такі аспекти:

- Орфографічні помилки
- Непрацюючі посилання
- Програма буде протестована на машинах з найнижчою специфікацією для тестування часу завантаження та будь-яких проблем із затримкою.

Бета-тестування - цей тест виконується після успішного виконання альфа-тестування. У бета-тестуванні зразок цільової аудиторії тестує програму. Бета-тестування також відоме як попереднє тестування. Бета-тестові версії програмного забезпечення ідеально розподіляються серед широкої аудиторії в Інтернеті, частково для того, щоб дати програмі «реальний» тест та частково забезпечити попередній перегляд наступного випуску. На цьому етапі аудиторія тестуватиме наступне:

- Користувачі встановлять, запустить програму та надішле свої відгуки команді проекту.
- Друкарські помилки, заплутаний потік додатків та навіть збої.
- Отримуючи відгуки, команда проекту може вирішити проблеми до випуску програмного забезпечення фактичних користувачів.
- Чим більше проблем ви вирішуєте для вирішення реальних проблем користувачів, тим вищою буде якість вашої програми.
- Наявність якіснішого додатка при його випуску для широкого загалу підвищить задоволеність клієнтів.

Нефункціональне тестування - цей розділ ґрунтується на тестуванні програми з його нефункціональних атрибутів. Нефункціональне тестування включає у собі тестування програмного забезпечення з вимог, які мають дисфункційний характер, але такі важливі, як продуктивність, безпека, інтерфейс користувача і т. буд.

Деякі з важливих і часто використовуваних дисфункцій тестування обговорюються нижче.

Тестування продуктивності - в основному використовується для виявлення будь-яких вузьких місць або проблем із продуктивністю, а не

пошуку помилок у програмному забезпеченні. Існують різні причини, які сприяють зниженню продуктивності програмного забезпечення:

- Затримка мережі
- Обробка на стороні клієнта
- Обробка транзакцій бази даних
- Балансування навантаження між серверами
- Передача даних

Тестування продуктивності вважається одним із важливих таобов'язкових типів тестування з погляду наступних аспектів:

- Швидкість (тобто час відгуку, передача даних та доступ)
- Місткість
- Стабільність
- Масштабованість

Тестування продуктивності може бути як якісним, так і кількісним і може бути поділено на різні підтипи, такі як тестування навантаження і стрес-тестування.

Тестування навантаження - це процес тестування поведінки програмного забезпечення шляхом застосування максимального навантаження з точки зору доступу до програмного забезпечення та управління великими вхідними даними. Це можна зробити як за нормального, так і в піковому навантаженні. Цей тип тестування визначає максимальну ємність програмного забезпечення та його поведінку у піковий час.

У більшості випадків тестування навантаження виконується за допомогою таких автоматизованих інструментів як Load Runner, AppLoader, IBM Rational Performance Tester, Apache JMeter, Silk Performer, Visual Load Load Test і т.д.

Віртуальні користувачі (VUsers) визначені в інструменті автоматичного тестування, і скрипт виконується для перевірки тестування навантаження програмного забезпечення. Кількість користувачів може збільшуватись або зменшуватись одночасно чи поступово залежно від вимог.

Стрес-тестування включає тестування поведінки програмного забезпечення у ненормальних умовах. Наприклад, це може включати видалення деяких ресурсів або застосування навантаження за межі фактичної межі навантаження.

Мета стрес-тестування полягає в тому, щоб протестувати програмне забезпечення, застосовуючи навантаження до системи та використовуючи ресурси, що використовуються програмним забезпеченням для визначення точки переривання. Це тестування може бути виконане шляхом тестування різних сценаріїв, таких як:

- Вимкнення або перезапуск мережних портів у випадковому порядку
- Увімкнення або вимкнення бази даних
- Запуск різних процесів, які споживають ресурси, такі як процесор, пам'ять, сервер тощо.

Тестування юзабіліті – це метод «чорної скриньки» та використовується для виявлення помилок та удосконалень програмного забезпечення шляхом спостереження користувачів за їх використання та роботу.

Згідно з Nielsen, юзабіліті можна визначити в термінах п'яти факторів, тобто ефективності використання, здатності до навчання, здатності пам'яті, помилок/безпеки та задоволеності. За його словами, зручність використання продукту буде гарною, і система може бути використана, якщо вона має вищевказані фактори.

Найджел Беван та Маклеод вважали, що зручність використання – це вимога якості, яку можна виміряти як результат взаємодії з комп'ютерною системою. Ця вимога може бути виконана і кінцевий користувач буде задоволений, якщо цільові цілі будуть ефективно досягнуті з використанням належних ресурсів.

У 2000 році Молич заявив, що зручна для користувача система повинна відповідати наступним п'яти цілям: легкість у навчанні, легко запам'ятовується, ефективна у використанні, задовільна та зручна у використанні.

На додаток до різних визначень зручності використання існують деякі стандарти та методи та методи, які визначають зручність використання у вигляді атрибутів та допоміжних атрибутів, таких як ISO-9126, ISO-9241-11, ISO-13407 та IEEE std. 610.12 і т.д.

Тестування інтерфейсу користувача включає в себе тестування графічного інтерфейсу Програмного забезпечення. Тестування інтерфейсу користувача гарантує, що GUI функціонує відповідно до вимог і перевіряється з точки зору кольору, вирівнювання, розміру та інших властивостей.

З іншого боку, тестування юзабіліті забезпечує зручний та зручний графічний інтерфейс, який можна легко обробляти. Тестування інтерфейсу користувача можна розглядати як частину перевірки юзабіліті.

Тестування безпеки включає тестування програмного забезпечення, щоб виявити недоліки і прогалини з точки зору безпеки і вразливості. Нижче наведено основні аспекти, які мають забезпечити тестування безпеки -

- Конфіденційність
- Цілісність
- Аутентифікація
- Доступність
- Авторизація
- Невідмовність
- Програмне забезпечення захищене від відомих та невідомихуразливостей
- Дані програмного забезпечення захищені
- Програмне забезпечення відповідає всім правилам безпеки
- Перевірка введення та перевірка
- Атаки вставки SQL
- Ін'єкційні дефекти
- Проблеми керування сеансом
- Міжсайтові скриптові атаки
- Буфер переповнює вразливість

- Атаки на обхід каталогу
- Тестування переносимості

Тестування переносимості включає тестування програмного забезпечення з метою забезпечення його повторного використання і можливість його перенесення з іншого програмного забезпечення. Нижче наведено стратегії, які можна використовувати для тестування переносимості:

- Перенесення програмного забезпечення з одного комп'ютера на інший.
- Побудова файлу, що виконується (.exe) для запуску програмного забезпечення на різних платформах [8].

Переносність тестування може розглядатися як одна з частин тестування системи, так як цей тип тестування включає загальне тестування програмного забезпечення щодо його використання в різних середовищах. Комп'ютерне обладнання, операційні системи та браузери є основним напрямом тестування переносимості. Деякі з попередніх умов для тестування переносимості полягають у наступному:

- Програмне забезпечення має бути спроектовано та закодовано з урахуванням вимог до переносимості.
- Модульні випробування виконувалися на зв'язаних компонентах.
- Виконано тестування інтеграції.
- Встановлено тестове середовище.

2.2. Модифікація методу для реалізації тестування backend API

Огляд аналогів показав, що існуючі реалізації є дуже складними в використанні та інтерфейс не сприяє швидкому отриманні даних з системи.

Тому важливо покращити можливість тестування додатків, модифікувавши метод тестування backend додатків додавши інтерфейс та покращивши методи роботи з CRUD. Складні системи та велика кількість фунціоналу не завжди добре, адже основна складність хорошого продукту - зробити його максимально зручним, швидким і зрозумілим. Тому було

вирішено розробити власний метод та програмний засіб для тестування backend частини додатків. Даний програмний засіб повинен містити простий дизайн з зрозумілим інтерфейсом та методом отримання і відправки даних, що дасть можливість швидко виконувати роботу з системою та модифіковувати систему під майбутні потреби.

Так як тестування включає в себе велику кількість різних типів і видів тестування - є важливим етапом виокремити конкретний тип тестування для якого і буде розроблено метод тестування. Попередня обробка вхідної інформації показала, що є сенс розглянути ручне тестування backend, так як автоматизоване тестування має іншу специфіку і не покриває тих аспектів якості, які покриває ручне тестування і що доступно розуму та винахідливості людини.

Тестування бекенд в сучасному світі відбувається, зазвичай, за допомогою API. API (Application Programming Interface) - набір готових класів, процедур, функцій, структур та констант, що надаються додатком (бібліотекою, сервісом) для використання у зовнішніх програмних продуктах [9].

Шар API будь-якої програми - один із найважливіших програмних компонентів системи. Це канал, який з'єднує клієнта із сервером (або один мікросервіс з іншим), керує бізнес-процесами та представляє сервіси, які приносять користь користувачам.

Загальнодоступний API, орієнтований на клієнта, який роблять відкритим для кінцевих користувачів, стає продуктом. Якщо він зламається, це ризикує не тільки один додаток, але й цілий ланцюжок бізнес-процесів, побудованих навколо нього.

Знаменита піраміда тестів Майка Кона поміщає тести API на сервісний рівень (інтеграційний), що передбачає, що близько 20% чи більше всіх наших тестів мають бути зосереджені лише на рівні API (точний відсоток залежить від потреб).

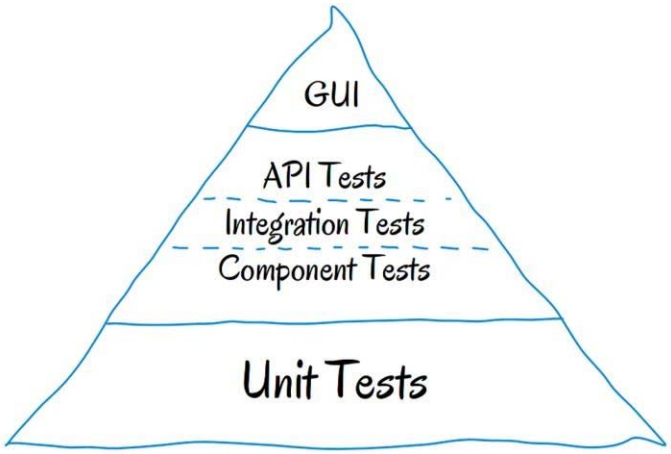


Рис. 2.1. Піраміда тестів

Коли ми вже маємо міцний фундамент із модульних тестів, що охоплюють окремі функції, тести API забезпечують більш високу надійність. Вони перевіряють інтерфейс, ближчий до користувача, але не мають недоліків тестів інтерфейсу користувача.

Тести API проходять швидко, забезпечують високу рентабельність інвестицій та спрощують перевірку бізнес-логіки, безпеки, відповідності та інших аспектів програми. У випадках, коли API є загальнодоступним, надаючи кінцевим користувачам програмний доступ до нашої програми або служб, тести API фактично стають наскрізними тестами і повинні охоплювати всю історію користувача (user story) [10].

В одному з аналогів тестування API полягає в виконанні багаторазово певних запитів при цьому навантажуючи сервіси чи сервери, інший ж аналог містить доволі складну структуру при налаштуванні. Великий функціонал зробив продукт дещо повільним.

У аналогах відсутні такі важливі фактори простота та швидкість роботи, тому, було прийнято рішення розробити метод для реалізації простого інтерфейсу на основі методу опрацювання даних.

Всі данні між клієнтом та сервером обмінюються за допомогою певного інтерфейсу (API). Данні надсилаються на сервер та отримуються з сервера за допомогою певних HTTP запитів. Ці запити виконуються на програмному рівні і користувач без знання мови програмування, що є частим явищем QA Engineer в компаніях, не зможе отримати інформацію про якість роботи backend частини

будь-якого додатку, а отже не може гарантувати якість продукту або ціна виявлення помилки на етапі GUI тестування (тестування графічного інтерфейсу, буде значно вища. Для невеликих компаній рзширити штат автоматизованими тестувальниками є велика розкіш, адже автоматизатори не зможуть витіснити або замінити ручних тестувальників повністю. Також виявлення дефектів на більш ранніх етапах – є основною задачою будь - якого тестувальника.

Основуючись на проблемах тестування backend частини додатків - було вирішено створити графічний інтерфейс, який за допомогою певного певних програмних рішень дозволить тестувальникам отримувати інформацію про стан речей на сервері, а також отримувати інформацію для тесту до моменту створення графічного інтрефейсу.

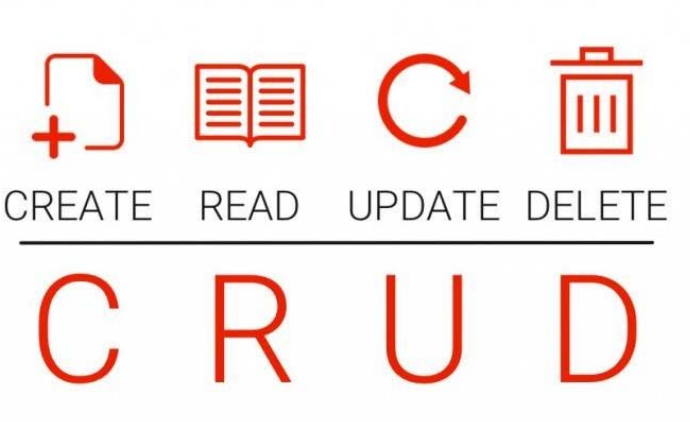
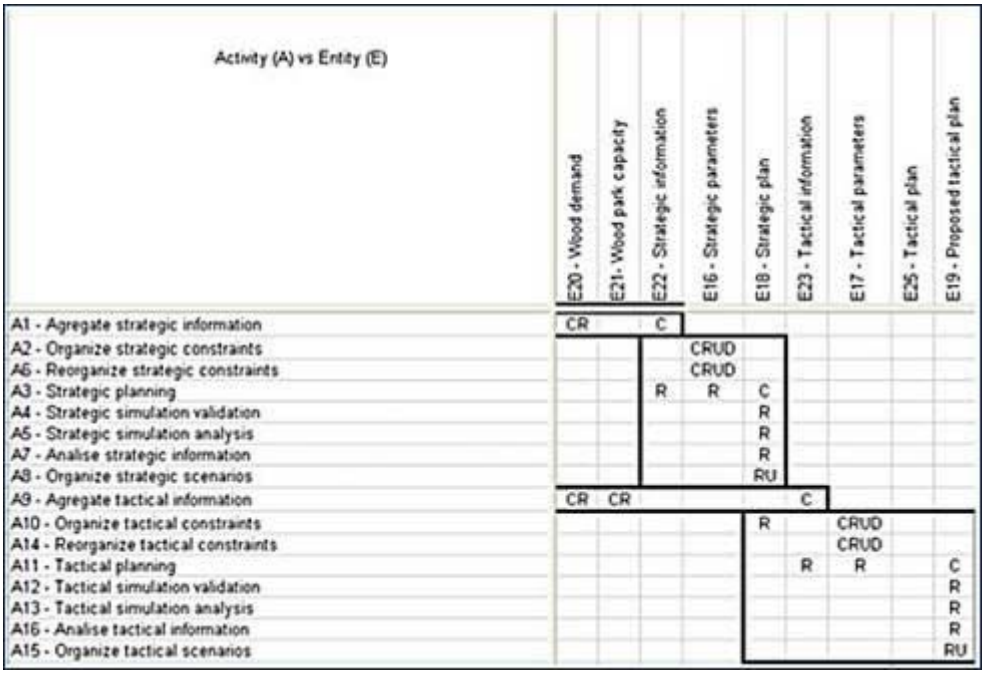


Рис. 2.2. Основні функції CRUD

Суть методу полягає в графічній реалізації CRUD методів роботи з даними. CRUD - акронім, що позначає чотири базові функції, що використовуються під час роботи з базами даних: створення (англ. create), читання (read), модифікація (update), видалення (delete). Введений Джеймсом Мартіном (англ. James Martin) у 1983 році як стандартна класифікація функцій з маніпуляції даними.

У SQL цим функціям, операціям відповідають оператори Insert (створення записів), Select (читання записів), Update (редагування записів), Delete (видалення записів). У деяких CASE-засобах використовувалися спеціалізовані CRUD-матриці (рис.2.3) або CRUD-діаграми, в яких для кожної сутності вказувалося, які базові функції з цією сутністю виконує той чи інший процес чи та чи інша роль. У системах, що реалізують доступ до бази даних через API у стилі REST, ці функції реалізуються часто (але не обов'язково) через HTTP-методи PUT, GET, PATCH та DELETE відповідно.



Activity (A) vs Entity (E)	E20 - Wood demand	E21- Wood park capacity	E22 - Strategic information	E16 - Strategic parameters	E18 - Strategic plan	E23 - Tactical information	E17 - Tactical parameters	E25 - Tactical plan	E19 - Proposed tactical plan
A1 - Agregate strategic information	CR		C						
A2 - Organize strategic constraints				CRUD					
A6 - Reorganize strategic constraints				CRUD					
A3 - Strategic planning			R	R	C				
A4 - Strategic simulation validation					R				
A5 - Strategic simulation analysis					R				
A7 - Analise strategic information					R				
A8 - Organize strategic scenarios					RU				
A9 - Agregate tactical information	CR	CR				C			
A10 - Organize tactical constraints					R		CRUD		
A14 - Reorganize tactical constraints							CRUD		
A11 - Tactical planning						R	R		C
A12 - Tactical simulation validation									R
A13 - Tactical simulation analysis									R
A16 - Analise tactical information									R
A15 - Organize tactical scenarios									RU

Рис. 2.3. Приклад CRUD-матриці

Метод, реалізований за допомогою програмного засобу дозволить не просто реалізовувати запити створення чи редагування для конкретної системи чи програмного додатку, а зробити майже універсальний додаток для тестування різного роду веб-додатків.

Так як CRUD включає в себе створення, читання, модифікацію і видалення даних - метод дозволяє вибрати тип дії, яку потрібно виконати. Для виконання цієї дії на певному сервері -потрібно отримати доступ або хоча б адресу цього сервера. Даний спосіб тестування передбачає поле, в яке можна

ввести URL серверу.

Для певного типу запитів, таких як створення чи модифікація, не достатньо самого лише типу запиту. Для створення нового користувача в системі, наприклад, потрібно вказати дані про цього користувача (ім’я, прізвище, вік, і так далі). Для передавання такої інформації - варто розмістити поле, де користувач ПЗ зможе вказати відповідні дані для окремого серверу чи сервіса з яким відбувається робота.

Так як самої можливості створення чи видалення інформації не достатньо для успішного тестування - необхідно розташувати елементи для отримання інформації, а саме для отримання статусу запита (код та текстове представлення статусу виконання запиту) і відповідь запиту (наприклад список студентів в групі чи книг в інтернет магазині).

2.3. Прототипування

У світі мільярди обчислювальних пристроїв. Ще більше програм для них. І кожен має свій інтерфейс, що є «важелями» взаємодії між користувачем і машинним кодом. Не дивно, що чим кращий інтерфейс, тим ефективніша взаємодія.

Однак далеко не всі розробники і навіть дизайнери замислюються про створення зручного і зрозумілого графічного інтерфейсу користувача. Є деякі основні принципи :

- Інтерфейс має бути інтуїтивно зрозумілим. Таким, щоб користувачеві не потрібно було пояснювати, як ним користуватися.
- Користувача краще повертати в місце, де він закінчив роботу минулого
- разу, щоб не потрібно було все те ж натискати знову.
- Найчастіше, користувачі в інтерфейсі спочатку шукають сутність (іменник), а потім дію (дієслово) до неї. Необхідно дотримуватись правила «іменник → дієслово». Наприклад, шрифт → змінити.
- Чим швидше людина побачить результат – тим краще. Приклад - "живий"

пошук, коли варіанти в процесі набору пошукового запиту. Основний принцип: програма має взаємодіяти з користувачем з урахуванням найменшої значимої одиниці вводу.

Перед створенням самого інтерфейсу ПЗ, необхідно зрозуміти, який інтерфейс необхідно створити:

- Просте має залишатися простим. Не варто ускладнювати інтерфейси. Постійно необхідно думати про те, як зробити інтерфейс простішим і зрозумілішим.
- Користувачі не замислюються над тим, як влаштовано програму. Все, що вони бачать – це інтерфейс. Тому, з погляду споживача, саме інтерфейс є кінцевим продуктом.
- Інтерфейс має бути орієнтованим на людину, тобто відповідати потребам людини та враховувати її слабкості. Потрібно постійно думати про те, з якими труднощами може мати справу користувач [11].

Розглянувши основні вимоги до інтерфейсу, можна чітко прослідкувати, що інтерфейс в першу чергу конструюється під потреби користувача і повинен бути максимально простим і зрозумілим. Саме такою буде першочергова ціль розробки інтерфейсу ПЗ.

Для створення максимально зручного і простого інтерфейсу необхідно встановити як саме буде проходити процес користування додатком. Цей процес багато в чому схожий на процес створення запиту в програмному коді, а саме вибір типу запиту, адреса на яку буде відбуватись запит, тіло запиту (за необхідності), та сама відправка. Крім відправки даних, ще необхідно отримати дані, тому останнім елементом інтерфейсу буде блок отримання відповіді та статусу запиту.

Так як програмний засіб розрахований на використання серед користувачів, де процес сприймання інформації починається зліва на право - є логічним розташувати перший елемент в лівій верхній частині прикладного інтерфейсу. По центру варто розташувати поле для введення адреси , так як це може бути доволі об’ємна інформація в одну строку. По праву сторону варто

розташувати кнопку відправки даних, так як тіло запиту в деяких запитах не обов'язково. Нижче буде розташовано блоки з тілом запиту і відповіддю від сервера. Також, як і у будь-якого ПЗ, необхідно додати назву програмного продукту та інформацію про несанкціоноване використання. Вважається доцільним розташовувати назву продукту на видноті для кращого запам'ятовування бренду, тому назва буде розташована по центру з самого верху інтерфейсу. Інформація про використання та автора ПЗ несе в собі більш інформаційний чи рекомендаційний характер, тому є логічним не акцентувати увагу користувача на цьому блоці і розташувати його внизу інтерфейсу.

Відповідно до вище зазначених вимог створено прототип інтерфейсу (рис. 2.4):

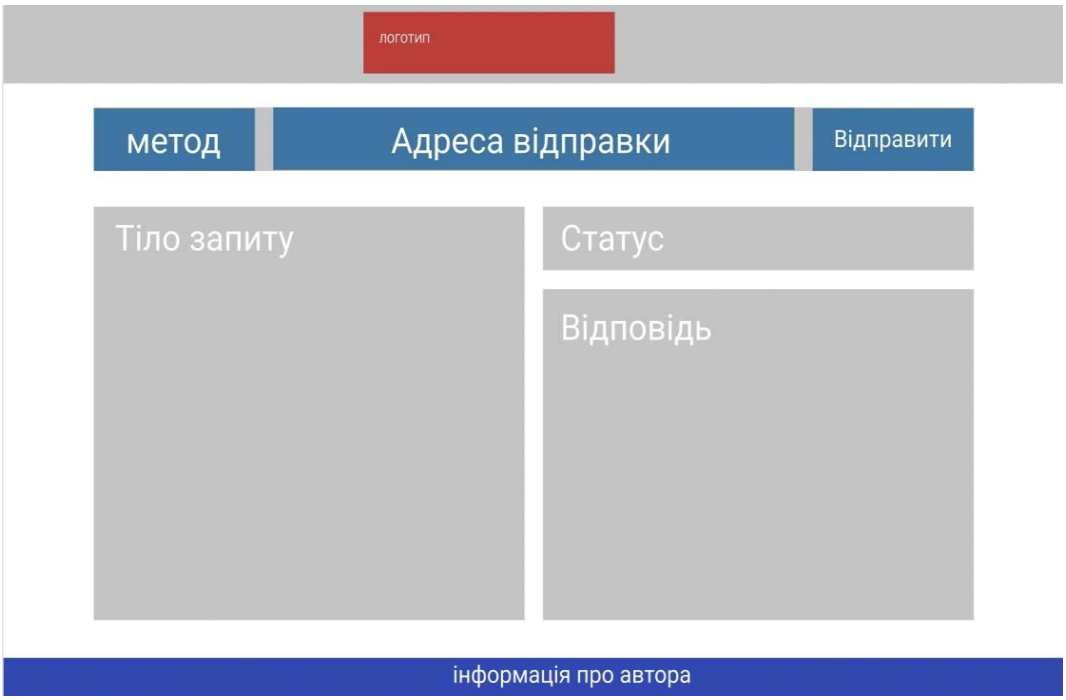


Рис.2.4. Прототип інтерфейсу програмного засобу

На основі цього прототипу розроблено дизайн-макет майбутнього додатку (рис. 2.5).

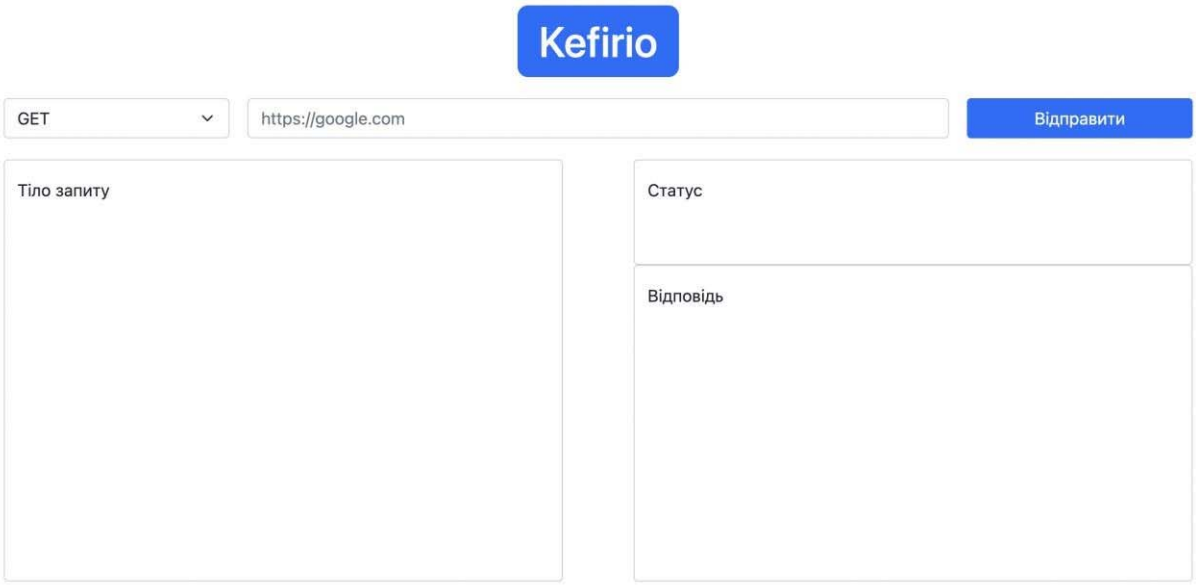


Рис. 2.5. Дизайн-макет ПЗ

Висновки до розділу II

У другому розділі кваліфікаційної роботи було проаналізовано методи та принципи тестування веб-додатків.

Було проаналізовано принципи тестування backend частини додатків, принцип обміну даних та основних потреб при тестуванні будь-якого додатку. Сформовано основні вимоги до методу та програмного засобу. Розроблено метод для реалізації програмного засобу для тестування backend частини за допомогою API через клієнтський інтерфейс.

Проаналізовано принципи створення інтерфейсів, внаслідок чого, було визначено основні потреби користувача і розроблено прототип майбутнього додатку. На основі прототипу розроблено дизайн-макет тобто інтерфейс програмного засобу.

РОЗДІЛ ІІІ
ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ТЕСТУВАННЯ BACKEND ЧАСНИНИ ВЕБ-ДОДАТКІВ

3.1. Обґрунтування вибору засобів для реалізації програмного засобу

Перед початком створення будь-якого програмного забезпечення - необхідно визначитись з ресурсами та технологіями, які будуть використовуватись для розробки ПЗ. В першу чергу необхідно визначитись з платформою, на якій буде розроблятись програмний засіб. Основними платформами є:

- Desktop application
- Web application
- Mobile

Розробка програмного забезпечення розпочалася з настільних програм, які можна було використовувати лише на автономних машинах. Однак з появою Інтернету та онлайн-комерції розробка веб-додатків набула значення. За визначенням, Desktop application означає будь-яке програмне забезпечення, яке можна встановити на одному комп'ютері (ноутбук або настільний комп'ютер) і використовувати для виконання конкретних завдань. Деякі настільні програми також можуть використовуватися кількома користувачами в мережевому середовищі [12].

Розробка desktop application включає в себе розробку багатьох складових для можливості розвертання додатків на ПК. Недоліками desktop application є:

- Необхідність встановлення додатку на ПК
- Чутливість до ОС
- Необхідність постійного оновлення додатків (автоматично або вручному режимі
- Велика ресурсозатратність при розробці

Розробка веб-додатків почала замінювати настільні програми з міркувань

портативності та кращих функцій з точки зору зручності використання. Розробка веб-додатків зазвичай виконується на архітектурі клієнт-сервер і використовує веб-браузер як клієнтський інтерфейс. Це одна з причин, чому веб-додатки так широко стають популярними. Хоча веб-програми мають певну перевагу перед настільними програмами, існує дуже мала ймовірність того, що настільні програми застаріють.

Основною причиною цього можуть бути проблеми безпеки та законності, пов’язані з веб-додатками.

Нижче наведено основне порівняння настільних і веб-додатків на основі певних параметрів:

Технічне обслуговування – веб-додатки необхідно встановлювати лише один раз, оскільки програми для настільних комп’ютерів встановлюються окремо на кожному комп’ютері. Крім того, оновлення програм є громіздким із настільними програмами, оскільки це потрібно робити на кожному окремому комп’ютері, що не стосується веб-програм.

Простота використання - настільні додатки обмежені фізичним розташуванням і, отже, мають обмеження щодо зручності використання. З іншого боку, розробка веб-додатків робить користувачам зручним доступ до програми з будь-якого місця за допомогою Інтернету.

Мобільні додатки - додатки, які встановлюються на мобільні пристрої такі як телефони, планшети та інші. Розробка мобільних додатків включає побудову архітектури, яка залежить від ОС пристрою, на якому буде використовуватиметься мобільний додаток, та апаратних можливостей пристроїв. Розробка ПЗ данної магістерської роботи можна вважати не доцільною через малі апаратні можливості пристроїв та не зручність використання мобільних пристроїв для тестування backend частини додатків.

Так як в данній роботі розглядається метод та програмний засіб для тестування API веб-додатків, є доцільним розробити саме веб-додаток, адже веб-додатки дають максимально зручність для користувача (можна використовувати перейшовши за посиланням), та не вимагає великої кількості

ресурсів як для розробки так і для наступного користування.

Розрізняють декілька типів веб-додатків:

- Односторінкові
- Багатосторінкові

Звичайний сайт складається з безлічі HTML-сторінок. При натисканні на посилання, браузер завантажує нові сторінки за цими посиланнями, з'являється відчуття руху від однієї сторінки до іншої. Сторінки можуть лежати як файли на якомусь сервері або генеруватися під запит якоюсь серверною програмою. Але, умовно кажучи, кожен екран сайту — це окрема технічна сутність, окремий документ. І користувач між ними рухається.

Мобільні програми, навпаки, ніби стоять на місці. У них завантажуються дані, змінюються екрани, але як результат - маємо відчуття, що ми завжди всередині цієї програми.

На початку 2010-х з'явилася нова концепція — щось середнє між сайтом та програмою. Таку архітектуру називають SPA - Single Page Application, наприклад Facebook.com.

SPA працює так: коли користувач відкриває сторінку, браузер завантажує одразу весь код програми. Але показує лише конкретний модуль - частина сайту, яка потрібна користувачеві. Коли користувач переходить в іншу частину програми, браузер бере вже завантажені дані та показує йому. І якщо потрібно, динамічно підвантажує з сервера потрібний контент без оновлення сторінки.

З одного боку, такі програми працюють швидко і менше навантажують сервер. З іншого боку, вони потребують більшого завантаження на старті.

Багатосторінковий сайт – це сукупність веб-сторінок, оформлених в одному стилі та об'єднаних спільною концепцією. Всі вони мають унікальні адреси, але масив даних пов'язаний доменним ім'ям. Користувач сприймає їх як єдине ціле. Є можливість переходу між сторінками одного сайту.

Існують величезні сайти. Число їхніх сторінок виражається п'ятизначними цифрами. Це найскладніші системи з особливими правилами

підпорядкування різним завданням. Для них створюються спеціальні

навігаційні карти. Але є інші форми існування в Інтернеті.

Як кожна людина має унікальний набір хромосом, так кожен ресурс має набір відмінних рис. До них входять:

- IP-адреса;
- Доменне ім'я;
- Назва сайту.

Кожна сторінка має власну URL-адресу. Це адреса, за якою вона знаходиться в мережі. За ключовими словами, пошукові машини розпізнають контент, розміщений на сторінках. Діючи за заданим алгоритмом, вони видають користувачам URL-сторінок, які можуть зацікавити їх. Порядковий номер у видачі визначає рівень популярності сайту.

Ресурс може належати приватній особі чи компанії. Немає офіційно затвердженого поділу на види чи типи сайтів. Але є ознаки, які дозволяють їх класифікувати [13].

Оскільки для розробки вибрано функціонал в мінімалістичному стиль, де навігація між ресурсами не потрібно, так як весь функціонал вміщується на одну сторінку і використовується почергово постійно. Тож проаналізувавши усі плюси і мінуси односторінкових і багатосторінкових веб-додатків, було прийнято рішення – розробити програмний засіб у стилі одно-сторінкового веб додатку.

Так як було вирішено розробляти веб-додаток, очевидно що розробку клієнтської частини додатку буде розроблено за допомогою HTML, CSS, Java Script.

Мова розмітки гіпертексту (HTML) — це набір символів або кодів розмітки, вставлених у файл, призначений для відображення в Інтернеті. Розмітка повідомляє веб-браузерам, як відображати слова та зображення веб-сторінки.

Кожен окремий код розмітки (який розташовується між символами "<" і ">") називається елементом, хоча багато людей також називають його тегом.

Деякі елементи представлені парами, які вказують, коли якийсь ефект

відображення має початися і коли він закінчиться.

Мова розмітки гіпертексту (HTML) — це основна мова сценаріїв, яку використовують веб-браузери для відтворення сторінок у всесвітній мережі.

Гіпертекст дозволяє користувачеві натиснути посилання та бути перенаправленим на нову сторінку, на яку посилається це посилання.

Ранні версії HTML були статичними (Web 1.0), тоді як новіші ітерації мають велику динамічну гнучкість (Web 2.0, 3.0).

Розмітка – це текст, який з'являється між двома загостреними дужками (наприклад, <виноска>), а вміст – це все інше.

Мова розмітки гіпертексту – це мова комп'ютера, яка полегшує створення веб-сайтів. Мова, яка має кодові слова та синтаксис, як і будь-яка інша мова, відносно проста для сприйняття і, з часом, стає все більш потужною в тому, що дозволяє комусь створювати. HTML продовжує розвиватися, щоб відповідати вимогам і вимогам Інтернету під прикриттям Консорціуму World Wide Web, організації, яка розробляє та підтримує мову; наприклад, з переходом на Web 2.0.

Гіпертекст – це метод, за допомогою якого користувачі Інтернету переміщуються в Інтернеті. Натискаючи на спеціальний текст, який називається гіперпосиланнями, користувачі переходять на нові сторінки. Використання гіпер означає, що він не є лінійним, тому користувачі можуть перейти в будь-яке місце в Інтернеті, просто натиснувши доступні посилання. Розмітка — це те, що HTML-теги роблять з текстом всередині них; вони позначають його як специфічний тип тексту. Наприклад, текст розмітки може бути виділений жирним шрифтом або курсивом, щоб привернути особливу увагу до слова чи фрази.

За своєю суттю HTML - це серія коротких кодів, введених у текстовий файл. Це теги, які забезпечують можливості HTML. Текст зберігається як файл HTML і переглядається через веб-браузер. Браузер зчитує файл і перекладає текст у видиму форму відповідно до кодів, які автор використав для написання того, що стає видимим відображенням. Написання HTML вимагає правильного

використання тегів для створення бачення автора.

Теги відрізняють звичайний текст від HTML-коду. Теги – це слова між так званими кутовими дужками, які дозволяють відображати графіку, зображення та таблиці на веб-сторінці. Різні теги виконують різні функції. Найпростіші теги застосовують форматування до тексту. Оскільки веб-інтерфейс має стати більш динамічним, можна використовувати каскадні таблиці стилів (CSS) і програми JavaScript. CSS робить веб-сторінки доступнішими, а JavaScript додає потужності базовому HTML.

На відміну від HTML, Extensible Markup Language, або XML, дозволяє користувачам визначати власну розмітку. Наприклад, використовуючи XML, один користувач може позначити виноску тегом <footnote>, а інший користувач може вибрати <fn>.

Використовуючи HTML, лише один заздалегідь визначений тег може використовуватися для позначення конкретного типу інформації. Документи XML призначені для легкого читання, оскільки вони містять визначені користувачем теги і оскільки документи складаються лише з розмітки та вмісту [14].

CSS (каскадні таблиці стилів) дозволяє створювати чудово виглядають веб-сторінки.

За допомогою CSS можна точно контролювати, як елементи HTML виглядають у браузері, представляючи свою розмітку, використовуючи будь-який дизайн, який подобається.

CSS — це мова для визначення того, як документи представляються користувачам — як вони оформлені, оформлені тощо. Документ зазвичай є текстовим файлом, структурованим за допомогою мови розмітки — HTML

Презентувати документ користувачеві означає перетворити його у форму, яку може використовувати аудиторія. Браузери, такі як Firefox, Chrome або Edge, призначені для візуального представлення документів, наприклад, на екрані комп’ютера, проектора чи принтера. CSS можна використовувати для дуже простих стилів тексту документа — наприклад, для зміни кольору та

розміру заголовків і посилань. Його можна використовувати для створення макета — наприклад, для перетворення окремого стовпця тексту в макет із основною областю вмісту та бічної панеллю для пов’язаної інформації. Його навіть можна використовувати для таких ефектів, як анімація.

CSS — це мова, заснована на правилах — визначаються правила, що визначають групи стилів, які слід застосовувати до певних елементів або груп елементів на вашій веб-сторінці. Наприклад, «Я хочу, щоб основний заголовок на моїй сторінці відображався великим червоним текстом» [15].

У наступному коді показано дуже просте правило CSS, яке дозволить досягти стилю, описаного вище:

```
h1 {
color: red;
font-size: 5em;
}
```

JavaScript — це текстова мова програмування, яка використовується як на стороні клієнта, так і на стороні сервера, що дозволяє зробити веб-сторінки інтерактивними. Якщо HTML і CSS є мовами, які надають структуру і стиль веб-сторінкам, JavaScript надає веб-сторінкам інтерактивні елементи, які залучають користувача. Поширені приклади JavaScript, які можуf використовувати щодня, включають вікно пошуку на Amazon, відео з підсумком новин, вбудоване в The New York Times, або оновлення стрічки Twitter.

Включення JavaScript покращує роботу веб-сторінки, перетворюючи її зі статичної сторінки в інтерактивну,так як JavaScript додає поведінку веб-сторінкам.

JavaScript в основному використовується для веб-додатків і веб-браузерів. Але JavaScript також використовується за межами Інтернету в програмному забезпеченні, серверах та вбудованих апаратних елементах керування.

Ось деякі основні речі, для яких використовується JavaScript:

1. Додавання інтерактивної поведінки на веб-сторінки. JavaScript дозволяє

користувачам взаємодіяти з веб-сторінками. Практично немає обмежень для того, що можна робити за допомогою JavaScript на веб-сторінці – це лише кілька прикладів:

- Показати або приховати додаткову інформацію одним натисканням кнопки
- Змініть колір кнопки, коли миша наведе на неї курсор
- Прокрутіть карусель із зображеннями на домашній сторінці
- Збільшення або зменшення масштабу зображення
- Відображення таймера або зворотного відліку на веб-сайті
- Відтворення аудіо та відео на веб-сторінці
- Відображення анімації
- Використання спадного меню гамбургерів

2. Створення веб- та мобільних додатків. Розробники можуть використовувати різні фреймворки JavaScript для розробки та створення веб- та мобільних додатків. Фреймворки JavaScript — це колекції бібліотек коду JavaScript, які надають розробникам попередньо написаний код для використання для рутинних функцій і завдань програмування — буквально каркас для створення веб-сайтів або веб-додатків.

Популярні інтерфейсні фреймворки JavaScript включають React, React Native, Angular і Vue. Багато компаній використовують Node.js, середовище виконання JavaScript, побудоване на движку JavaScript V8 Google Chrome. Кілька відомих прикладів включають Paypal, LinkedIn, Netflix та Uber.

3. Створення веб-серверів і розробка серверних додатків. Крім веб-сайтів і програм, розробники також можуть використовувати JavaScript для створення простих веб-серверів і розвитку внутрішньої інфраструктури за допомогою Node.js.
4. Розробка гри. Звичайно, також можна використовувати JavaScript для створення браузерних ігор. Це чудовий спосіб для початківців розробниківпрактикувати свої навички JavaScript.

Окрім необмежених можливостей, для веб-розробників є багато причин використовувати JavaScript замість інших мов програмування:

- JavaScript є єдиною мовою програмування, яка є рідною для веб-браузера
- JavaScript є найпопулярнішою мовою
- Для початку існує низький поріг
- Це цікава мова для вивчення

Дані мови розмітки та програмування вибранні через їх унікальну можливість будувати гнучкі веб-сторінки та через те, що саме ці структурні мови були базовими при створенні веб-сайті на початку їх виникнення та розвивались з розвитком Інтернету. JavaScript вибрано через можливість швидко та динамічно впливати на сторінку веб-сайту навіть без перезавантаження її. Також JavaScript було вибрано через можливість масштабування додатку за рахунок написання не тільки клієнтської частини на цій мові програмування, а і серверну на її фрейморках, таких як Node.js.

Мова JavaScript не має власного середовища виконання, а тому є кросплатформеною і чудово підходить для виконання на сервері.

Node або Node.js - програмна платформа, заснована на двигуні V8 (що транслює JavaScript в машинний код), що перетворює JavaScript з вузькоспеціалізованої мови на мову загального призначення. Node.js додає можливість JavaScript взаємодіяти з пристроями вводу-виводу через свій API, написаний на C++, підключати інші зовнішні бібліотеки, написані різними мовами, забезпечуючи виклики до них із JavaScript-коду. Node.js застосовується переважно на сервері, виконуючи роль веб-сервера, але є можливість розробляти на Node.js та десктопні віконні програми (за допомогою NW.js, AppJS або Electron для Linux, Windows та macOS) і навіть програмувати мікроконтролери (наприклад, tessel, low.js та espruino). В основі Node.js лежить подієво-орієнтоване та асинхронне (абореактивне) програмування з неблокуючим введенням/виводом.

Порівняльний аналіз найпопулярніших мов програмування, які

дозволяють розробляти серверну частину додатку наведено у таблиці 3.1.

Таблиця 3.1 – Порівняльний аналіз серверних мов програмування

Критерії	Node.js	Java	PHP
Простота у використанні	+	-	+
Підтримка асинхронності	+	+	+
Потужний набір інструментів розробника	+	+	-
Швидкість виконання	+	+	-

З огляду на порівняльний аналіз серверних мов програмування, зрозуміло, що Node.js є найкращим вибором для розробки серверної частини веб додатку. Окрім ряду переваг над аналогами, мова програмування що використовується для розробки клієнтської та серверної частин є однаковою, а саме – JavaScript, що значно спрощує процес розробки. Серверна частина додатку потрібна для побудови backend для тестування самого сервісу.

Після вибору мови програмування, за допомогою якої буде виконуватись розробка програмного засобу, необхідно вибрати IDE.

Інтегроване середовище розробки, ICP (англ. Integrated development environment - IDE), також єдине середовище розробки, ECP - комплекс програмних засобів, що використовується програмістами для розробки програмного забезпечення (ПЗ).

Середовище розробки включає:

- текстовий редактор,
- транслятор (компілятор та/або інтерпретатор),
- засоби автоматизації складання,
- відладчик.

Іноді містить також засоби для інтеграції із системами керування версіями та різноманітні інструменти для спрощення конструювання графічного інтерфейсу користувача. Багато сучасних середовищ розробки також включають браузер класів, інспектор об'єктів та діаграму ієрархії класів - для використання при об'єктно-орієнтованій розробці ПЗ. ICP зазвичай призначені для кількох мов програмування - такі як IntelliJ IDEA, NetBeans, Eclipse, Qt Creator, Geany, Embarcadero RAD Studio, Code::Blocks, Xcode або Microsoft Visual Studio, але є IDE для однієї певної мови програмування - як, наприклад , Visual Basic, Delphi, Dev-C ++.

Для розробки веб-додатків зачасту використовують Visual Studio Code для створення клієнтської та серверної частини додатків. Зручний інтерфейс, можливість налаштування програми під свої потреби через встанновлення великої кількості плагінів та досить зручний термінал для роботи з сервером та системами контролю версії - стали визначними показниками при виборі IDE. Так як VS Code містить всі ці параметри - саме це середовище розробки було вибрано для розробки данного програмного засобу.

3.2. Особливості розробки програмного забезпечення

Напевно, одне із найважливіших запитань, яке виникає на початку розробки будь-якого програмного засобу є – архітектура, тому що від того, на скільки правильно та продумано спроектована архітектура - залежить уся наступна розробка. Чим краще підібрана та розроблена архітекутра, тим легше в майбутньому виконується розробка ПЗ.

Програмний засіб, що розробляється, є веб-додатком, що складається з клієнта, який забезпечує взаємодію між користувачем та сервером, на якому відбувається логіка роботи сервісу для роботи з данними та взаємодія з данними, де розміщується інформація. Виділяються декілька видів архітектури, кожна краще підходить для вирішення певної задачі, з певними плюсами та мінусами. Також буває, що стилі архітектури комбінуються в

межах розробки одного програмного продукту, задля розробки кращого архітектурного рішення. Найчастіше вирізнають монолітну та мікросервісну стилі архітектури

Монолітний додаток (моноліт) - додаток, що доставляється через єдине розгортання. Таким є додаток, доставлений у вигляді однієї WAR або додаток Node з однією точкою входу.

Прикладом є класичний інтернет-магазин. Стандартні модулі: UI, бізнес-логіка та data-шар. Можливі способи взаємодії із сервісом: API REST та веб-інтерфейс.

При побудові моноліту всі ці речі керуватимуться всередині того самого модуля.

На рисунку 3.1 відображено всі частини додатку, які знаходяться в тому самому модулі розгортання:

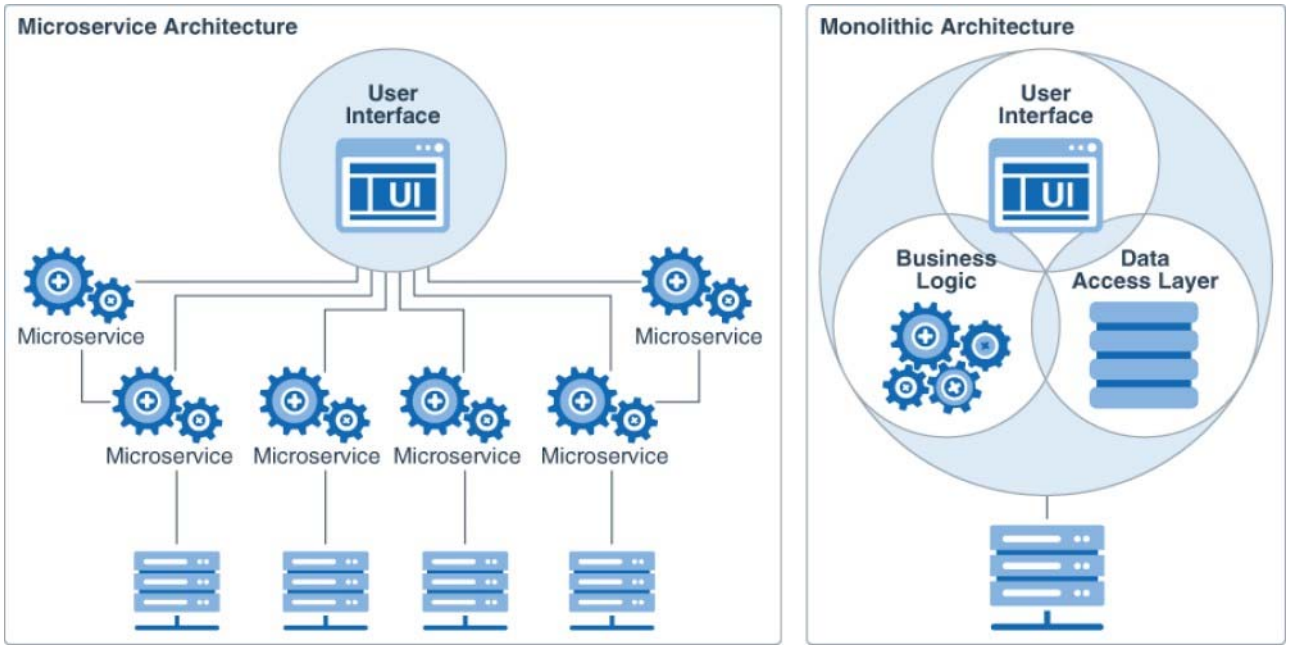


Рис.3.1. Схема архітектурних типів

Великою перевагою моноліту є те, що його легше реалізувати. У монолітній архітектурі можна швидко почати реалізовувати свою бізнес-логіку замість того, щоб витрачати час на роздуми про міжпроцесну взаємодію.

Ще одна річ – це наскрізні (E2E) тести. У монолітній архітектурі їх легко виконати.

Говорячи про операції, важливо сказати, що моноліт простий у розгортанні і легко масштабується. Для розгортання ви можете використовувати скрипт, що завантажує ваш модуль і запускає програму. Масштабування досягається шляхом розміщення Loadbalancer перед кількома екземплярами програми. Як ми можемо бачити, моноліт є досить простим в експлуатації.

Моноліти, як правило, перероджуються зі свого чистого стану в так звану «велику кульку бруду». Коротко це описується як стан, який виник, тому що архітектурні правила були порушені і з часом компоненти зрослися.

Це переродження уповільнює процес розробки: кожну майбутню функцію складніше розвиватиме. Через те, що компоненти ростуть разом, їх також необхідно міняти разом. Створення нової функції може означати дотик до 5 різних місць: 5 місць, де потрібно написати тести; 5 місць, які можуть мати небажані побічні ефекти для наявних функцій.

Моноліт легко масштабувати доти, доки він не переросте у «велику кульку бруду», як згадувалося раніше. Масштабування може бути проблематичним, коли лише одній частині системи потрібні додаткові ресурси, адже у монолітній архітектурі ви не можете масштабувати окремі частини вашої системи.

У моноліті практично немає ізоляції. Проблема або помилка в модулі може уповільнити або зруйнувати всю програму.

Будівництво моноліту часто відбувається за допомогою вибору основи. Вимкнення або оновлення початкового вибору може бути скрутним, тому що це має бути зроблено відразу і для всіх частин вашої системи.

У свою чергу у мікросервісній архітектурі слабко пов'язані сервіси взаємодіють один з одним до виконання завдань, які стосуються їх бізнес-можливостей.

Мікросервіси значною мірою отримали свою назву через те, що сервіси тут менше, ніж у монолітному середовищі. Проте мікро — про бізнес-можливості, а не про розмір.

Порівняно з монолітом у мікросервісах є кілька одиниць розгортання. Кожен сервіс розгортається самостійно.

Для прикладу знову розглянемо Інтернет-магазин. Як і раніше, у маємо: UI, бізнес-логіка та data-шар.

Тут відмінність від моноліту у тому, що в усіх перелічених вище є свій сервіс і своя база даних. Вони слабко пов'язані і можуть взаємодіяти з різними протоколами (наприклад, REST, gRPC, обмін повідомленнями) через межі.

Перавагою мікросервісів є те, мікросервіси легше тримати модульними. Технічно це забезпечується твердими межами між окремими сервісами.

У великих компаніях різні послуги можуть належати різним командам. Послуги можуть бути використані всією компанією. Це також дозволяє командам працювати над послугами переважно самостійно. Немає потреби координувати розгортання між командами. Розвивати послуги краще зі збільшенням кількості команд.

Мікросервіси менші, і завдяки цьому їх легше зрозуміти та перевірити.

Менші розміри допомагають, коли йдеться про час компіляції, час запуску та час, необхідний для виконання тестів. Всі ці фактори впливають на продуктивність розробника, тому що дозволяють витрачати менше часу на очікування на кожному етапі розробки.

Коротший час запуску та можливість розгортання мікросервісів незалежно один від одного дійсно вигідні для CI/CD. Порівняно із звичайним монолітом він набагато плавніший.

Мікросервіси не прив'язані до технології, яка використовується в інших сервісах. Значить ми можемо використовувати кращі технології припасування.

Старі послуги можуть бути швидко переписані для використання нових технологій.

У мікросервісах ізольовані розломи краще порівняно з монолітним підходом. Добре спроектована розподілена система переживе збій одного сервісу.

Все звучить досить добре, але є недоліки. Розподілена система має свою

складність: у ній вам доводиться мати справу з частковою відмовою, більш складною взаємодією при тестуванні (тести E2E), а також з більш високою складністю при реалізації взаємодії між сервісами.

Транзакції легше проводити у моноліті. Вирішенням цієї проблеми на мікросервісах є Saga Pattern. Хороше рішення, але все ж таки надто громіздке для реалізації на практиці.

Існують експлуатаційні накладні витрати, а безліч мікросервісів складніше в експлуатації, ніж кілька екземплярів сигнального моноліту.

Крім перерахованих вище складнощів, для мікросервісів також може знадобитися більше обладнання, ніж для традиційних монолітів. Іноді мікросервіси можуть перевершити один моноліт, якщо є його частини, які вимагають масштабування до краю.

Зміни, що стосуються кількох сервісів, повинні координуватися між кількома командами, а це може бути складно, якщо команди ще не мали контактів [16].

Провівши аналіз архітектури, було вибрано монолітну архітектуру через просто реалізацію, менші затрати на реалізацію та підтримку проекту, що є важливим при можливій монетизації в майбутньому.

Робота з будь-яким додатком починається з відкриття, тобто, за допомогою браузера, користувач в адресний рядок вводить адресу веб-додатку, після чого відбувається порівняння за системою DNS доменного імені з IP

комп’ютера, на якому розташований ресурс. Після цього користувач отримує початкову сторінку, де він може взаємодіяти з продуктом. Загальний алгоритм роботи програмного засобу для реалізації ігрових інтерактивних дій зображено на рисунку 3.2.

Рис. 3.2. Загальний алгоритм роботи програмного забезпечення

Спершу користувач відриває додаток за певною адресою, після чого вибирає тип запиту, який потрібно, ввести URL за яким потрібно виконати запит та внести додаткові данні в тілі запиту. Після цього відбувається обробка данних та відправка на сервер. Після виконання сервером запиту - повертається відповідь з тілом відповіді та данними про запит. Якщо на стадії відправки запиту чи його отримання буде помилка - користувач отримає повідомлення

про збій чи не коректне введення даних.

API (application program interface) – програмний інтерфейс,який описує набір методів, за допомогою яких, певний програмний засіб має змогу взаємодіяти з іншим програмним засобом. Для взаємодії клієнта з сервером, було прийнято рішення розробити серверне API у стилі REST.

REST (від англ. Representational State Transfer - "передача репрезентативного стану" або "передача "самоописуваного" стану") - архітектурний стиль взаємодії компонентів розподіленого додатка в мережі. Іншими словами, REST - це набір правил про те, як програмісту організувати написання коду серверного додатка, щоб всі системи легко обмінювалися даними і додаток можна було масштабувати.

У певних випадках (інтернет-магазини, пошукові системи, інші системи, засновані на даних) це призводить до підвищення продуктивності та спрощення архітектури. У широкому значенні [уточнити] компоненти в REST взаємодіють на кшталт взаємодії клієнтів та серверів у Всесвітньому павутинні. Його роботу наведено на рисунку 3.3.

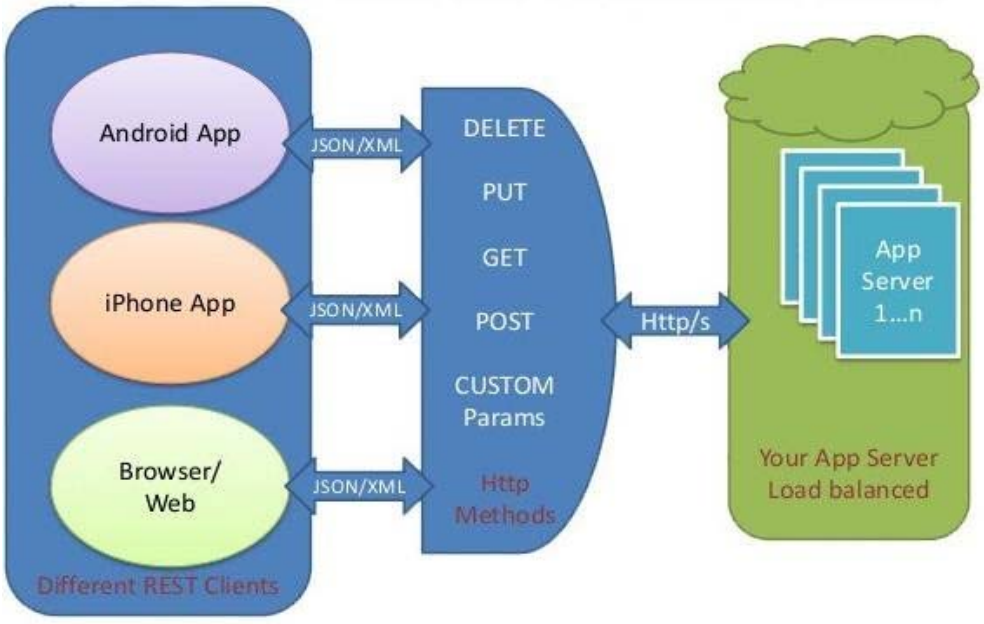


Рис. 3.3. Алгоритм роботи REST API

REST – один з найпопулярніших підходів створення API, так як він підкреслює простоту, розширюваність, надійність та продуктивність.

Властивості архітектури, які залежать від обмежень, накладених на REST-системи:

- Продуктивність - взаємодія компонентів системи може бути домінуючим фактором продуктивності та ефективності мережі з погляду користувача;
- Масштабованість для забезпечення великої кількості компонентів та взаємодій компонентів.
- Рой Філдінг - один з головних авторів специфікації протоколу HTTP, описує вплив архітектури REST на масштабованість таким чином:
- Простота уніфікованого інтерфейсу;
- Відкритість компонентів до можливих змін для задоволення потреб, що змінюються (навіть при працюючому додатку);
- Прозорість зв'язків між компонентами системи для сервісних служб;
- Перенесення компонентів системи шляхом переміщення програмного коду разом з даними;
- Надійність, що виражається у стійкості до відмов на рівні системи за наявності відмов окремих компонентів, сполук або даних.

REST API складається з так HTTP методів, кожен з них має своє призначення. Основними HTTP методами є POST, GET, PUT, і DELETE. Вони відповідають операціям створення, читання, оновлення та видалення.

- GET – використовується для отримання даних з сервера, це означає

«дати мені уявлення про цей ресурс, ідентифікованому з цього URL». Варто зазначити, що GET ніколи не варто використовувати для зміни ресурсів. Для цього є інші методи.

- POST – використовується для надання даних на сервер, який означає «обробляти уявлення ресурсу». На практиці POST використовується для створення ресурсів. Клієнт вказує уявлення ресурсу в тілі запиту і створює POST для URL-адреси, який містить колекцію ресурсів.
- PUT – використовується для оновлення існуючих ресурсів сервера

шляхом заміни старого стану на нове, що означає «використовувати це уявлення для заміни ресурсу, ідентифікованого цим URL».

- DELETE – використовується для видалення ресурсів сервера, що означає «видалити ресурс, зазначений у цій URL».

На рисунку 3.4 наведено фрагмент коду, що поєднує всі роути між собою.

```
const app = express();

const PORT = 5000;

app.use(cors());
app.use(bodyParser.json());

app.use('/users', usersRoutes);
```

Рис. 3.4. Поєднання роутів між собою

На рисунку 3.5 наведено фрагмент коду, що реалізує REST API для роботи з користувачем.

```
const router = express.Router();

router.get('/', getUsers);
router.post('/', createUSer);

router.get('/:id', getUser);

router.delete('/:id', deleteUser);

router.patch('/:id', updateUser);
export default router;
```

Рис. 3.5. Маршрутизація сутності “Користувач”

Однією з головних задач реалізації API, за який у відповідає створення – не змішувати бізнес логіку з рівнем створення. Для виконання бізнес логіки служить інший рівень. Таким чином, забезпечується багаторівнева архітектура.

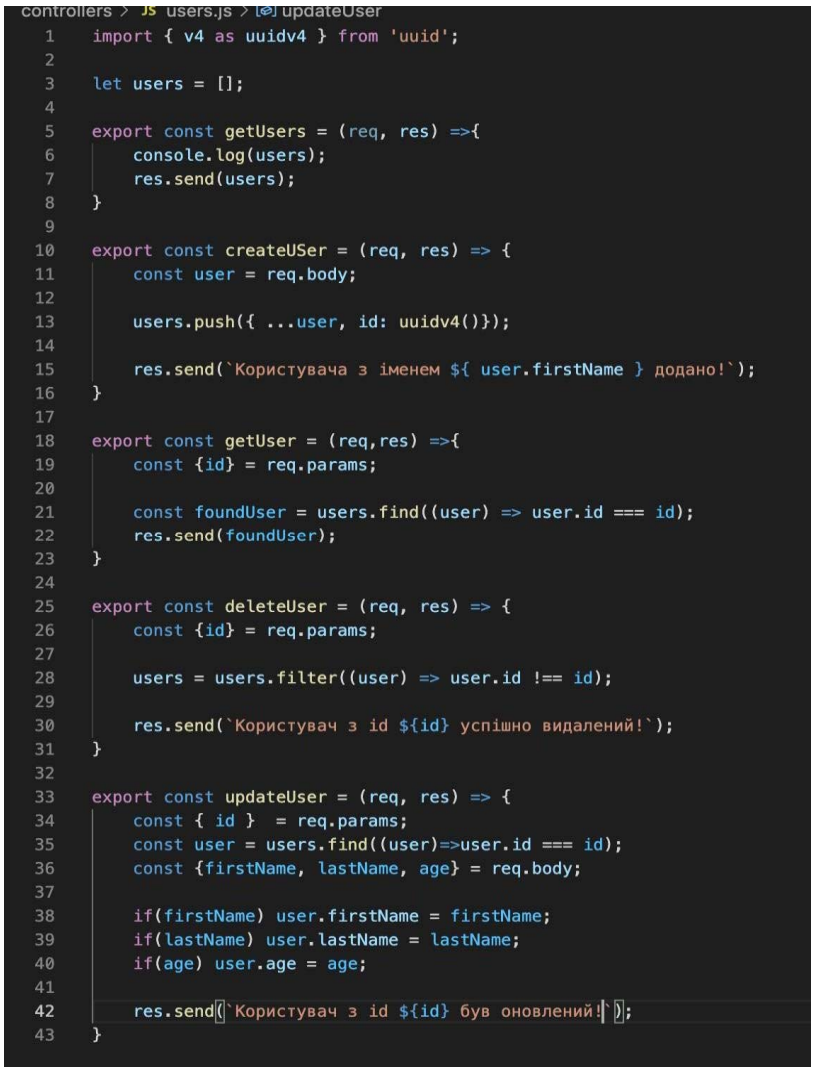


Рис.3.6. Контролери для керування сервером

Також було розроблено логіку контролерів серверної частини для виконання сервером певних маніпуляцій з користувачем. Приклад цієї частини наведено на рисунку 3.6.

Метод полягає в отриманні та зберіганні даних, які будуть використовуватись для відправки на сервер та подальшої обробки. Суть методу полягає в валідації полів відповідно до методу блокуванню полів, якщо для даного методу немає потреби вводити данні, збір та збереження даних в об’єкті, який і передається в запиті на сервер (рис. 3.7).

Користувач заходить на веб сторінку, вибирає необхідний метод, який він буде використовувати для тесту. Після такого вибору відбувається фільтрація який саме метод було вибрано і далі блокування поля тіла запиту, якщо це не потрібно для цього методу.

```
const output = document.getElementById("output"),
    select = document.getElementById("select"),
    basicUrl = document.getElementById("basic-url"),
    floatingTextareaIn = document.getElementById("floatingTextareaIn"),
    reqStatus = document.querySelector("#status"),
    submit = document.getElementById("submit");


    // Call function getDataFromFields if event is click
    submit.addEventListener('click', function (event){
        event.preventDefault();
      getDataFromFields(floatingTextareaIn,basicUrl,select)
    });

//block textarea if method is GET
select.addEventListener('change', function(){
  if (select.value === "Get"){
    floatingTextareaIn.setAttribute("readonly","readonly");
  } else  {
    if(floatingTextareaIn.hasAttribute("readonly")){
      floatingTextareaIn.removeAttribute("readonly");
    }
  }
});

  function getDataFromFields (inputArea, url,select){
    let info = {};
    info.inputArea = inputArea.value;
    info.url = url.value;
    info.select = select.value;
    console.log(info);
   chooseRequest(info);
}
```

Рис. 3.7. Метод отримання та збереження даних

Після введення всіх необхідних даних для запиту і натискання на кнопку відправки запиту, відбувається збереження в об’єкт, який і поєднує всю інформацію яка буде передаватись на серверну частину.

У веб-додатку було реалізовано методи GET, POST, DELETE, PATCH. Для кожного методу роботи з даними було розроблено свій метод в веб-додатку.

Для роботи з CRUD методами використано бібліотеку Axios.

Axios — це HTTP-клієнт на основі промісі для node.js і браузера. Він ізоморфний (може працювати у браузері та в nodejs з однаковою кодовою базою). На стороні сервера він використовує власний http-модуль node.js, тоді як на клієнті (браузері) він використовує XMLHttpRequests [17].

Ця бібліотека дає можливість простішим способом отримувати та

передавати дані між клієнтом та сервером. [18]

На рисунку 3.8 відображена реалізація методі отримання та запису даних, а на рисунку 3.9 методи редагування та видалення.

```
function getInfo(info){
    if(info.url === ''){
        alert("Вкажіть адресу запиту!");
    }else {
    axios.get(info.url, {
    })
    .then(function (response) {
        output.append(JSON.stringify(response.data,null,'\t'));
        console.log(response);
        reqStatus.append('status: '+response.status +'\n' + 'statusText: '+ response.statusText);

    })
    .catch(function (error) {
      console.log(error);
    })
    .then(function () {
      // always executed
    });
}}

function postInfo(info){
  if(info.url === ''){
    alert("Вкажіть адресу запиту!");
}else {
  const jsonObj = JSON.parse(info.inputArea);
    axios({
      method: "POST",
      data: jsonObj,
      url: info.url
  })
      .then(function (response) {
        swal("Успіх",response.data , "success");
        console.log(response);
      })
      .catch(function (error) {
        console.log(error);
      });
}}
```

Рис. 3.8. Реалізація методів отримання та запису

Метод getInfo() отримує дані з заданою адресою, якщо поле адреси не пусте. Відбувається запит на сервер додатку та отримання інформації від сервера. Якщо підчас запиту на сервер були неполадки – буде виведена помилка.

В свою чергу метод postInfo() виконується при записі на сервер даних та записує дані, які були вказані в полях раніше за адресою, якщо вона не пуста. Якщо адреса серверу пуста – буде видана помилка. При успішному виконанні буде відображено модальне вікно з інформацією про успішну дію.

```
function patchInfo(info) {
  console.log("patch");
  if(info.url === ''){
    alert("Вкажіть адресу запиту!");
}else {
  const jsonObj = JSON.parse(info.inputArea);
    axios({
      method: "PATCH",
      data: jsonObj,
      url: info.url
  })
      .then(function (response) {
        swal("Успіх",response.data , "success");
        console.log(response);
      })
      .catch(function (error) {
        console.log(error);
      });
}}

function deleteUser(info) {
  console.log("delete");
  if(info.url === ''){
    alert("Вкажіть адресу запиту!");
}else {

    axios({
      method: "DELETE",

      url: info.url
  })
      .then(function (response) {
        swal("Успіх",response.data , "success");
        console.log(response);
      })
      .catch(function (error) {
        console.log(error);
      });
}}
```

Рис. 3.9. Реалізація методів редагування та видалення

Метод patchInfo() отримує в параметрі інформацію, яку було введено в поле «Тіло запит» та за адресою сервера та унікального ідентифікатору елемента в базі даних замінює вказані поля, не змінюючи інші поля. Такий метод дає можливість редагувати інформацію уже створених об’єктів.

Метод deleteUser() видаляє запис на сервері за унікальним ідентифікатором.

Дані методи дають можливість реалізувати базові методі взаємодії для з сервером через API по кліку на кнопку «Відправити».

3.3. Тестування програмного забезпечення

Тестування - один з найважливіших етапів розробки програмного забезпечення. Розробка програмного засобу розробником чи командою розробників включає в себе людський фактор, а значить не може гарантувати абсолютної якості та стабільності програмного засобу. В одному з принципів тестування вказано: «Тестування може показати наявність дефектів в програмі, але не довести їх відсутність». Однак, це не означає, що тестування не потрібно проводити, навпаки, це вказує на те, що процес тестування допоможе виявити максимальну кількість дефектів до моменту релізу (випуску на ринок) продукту. Тестування ділиться на багато видів в залежності від багатьох факторів. Ось декілька з них:

1. Функціональне

Функціональне тестування – один із видів тестування, спрямованого на перевірку відповідностей функціональних вимог ПЗ його реальним характеристикам. Основним завданням функціонального тестування є підтвердження того, що програмний продукт, який розробляється, володіє усім необхідним замовнику функціоналом.

2. Регресійне

Регресійне тестування – це набір тестів, що спрямовані на виявлення дефектів у вже протестованих модулях додатку. Робиться це зовсім не для того, щоб остаточно переконатися у відсутності багів, а для пошуку та виправлення регресійних помилок. Регресійні помилки – ті ж баги, але з'являються вони не при написанні програми, а при додаванні до існуючого білду нової частини програми або виправлення інших багів, що і стає причиною виникнення нових дефектів у вже протестованому продукті .

3. Ad hoc

Інтуїтивне тестування (ad-hoc testing) - вид тестування, який виконується без підготовки до тестів, без визначення очікуваних результатів, проектування тестових сценаріїв. Це неформальне імпровізаційне тестування. Він не вимагає

жодної документації, планування, процесів яких слід дотримуватися у виконанні. Також на даний вид тестування не пишуться тест-кейси, що, в свою чергу, може викликати певні труднощі у спробах відтворити дефект у системі. Такий вид часто може дати відразу більше результату ніж тестування за певними сценаріями. Це пов'язано з тим, що тестувальник на перших кроках приступає до тестування основного функціоналу та виконує нестандартні перевірки, точніше деякі з його перевірок будуть нестандартними.

4. Тестування сумісності. Даний вид тестування перевіряє чи розроблена гра буде правильно працювати в рамках встановлених вимог до системи. Через велику кількість пристроїв та платформ, гра перевіряється на кожній, де перевіряється забезпечення одноманітності гри в незалежності від версії чипристрою.
5. Навантажувальне тестування — підвид тестування продуктивності, збирання показників та визначення продуктивності та часу відгуку програмно- технічної системи або пристрою у відповідь на зовнішній запит з метою встановлення відповідності вимогам до цієї системи (пристрою).

Для дослідження часу відгуку системи на високих або пікових навантаженнях проводиться стрес-тестування, у якому створюване систему навантаження перевищує нормальні сценарії її використання. Не існує чіткої межі між навантажувальним та стрес-тестуванням, проте ці поняття не варто змішувати, тому що ці види тестування відповідають на різні бізнес-питання та використовують різну методологію.

Проте, усі види тестування діляться на два методи: методи тестування чорного та білого ящика.

Black Box Testing – це метод тестування програмного забезпечення, за допомогою якого функціональні можливості програмних додатків перевіряються без знання внутрішньої структури коду, деталей реалізації та внутрішніх шляхів. Тестування Black Box в основному зосереджується на введенні та виводі програмних додатків і повністю базується на вимогах і

специфікаціях програмного забезпечення. Він також відомий як поведінковий тест [19].

«Тестування білого ящика» (також відоме як тестування прозорої, скляної коробки або структурного тестування) — це техніка тестування, яка оцінює код і внутрішню структуру програми.

Тестування «білого ящика» передбачає перегляд структури коду. Коли знаємо внутрішню структуру продукту, можна провести випробування, щоб переконатися, що внутрішні операції виконуються відповідно до специфікації. І всі внутрішні компоненти відпрацьовані належним чином [20].

Тестування додатку було проведено на основі чорного ящика. Для цього, було відтворено основні можливі тестові сценарії та порівняння їх очікуваного та фактичного результату [21].

Робота додатку починається з запуску його в браузері та відображення інтерфейсу на екрані користувача (рисунок 3.10).

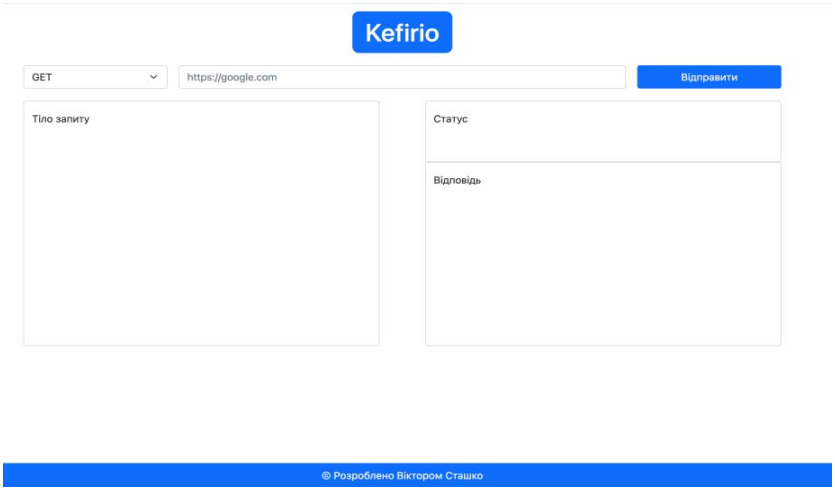


Рис. 3.10. Відображення інтерфейсу користувача

Після цього необхідно перевірити можливість вибору різного типу запитів. Для цього по натиску на випадаючий список в лівому верхньому краї повинні відображатись 4 варіантів запитів: Get, Post, Patch, Delete [22]. Результат роботи випадаючого списку на рисунках 3.11 і 3.12.

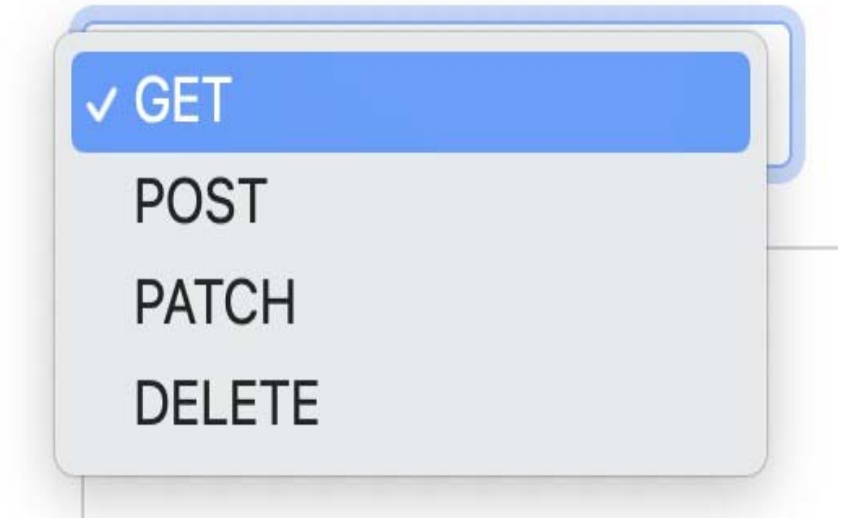


Рис..3.11. Варінти випадаючого списку

Рис..3.12. Результат зміни типу запиту

Наступною функцією даного додатку є можливість вводу данних в поля (адреси та тіла запиту), це означає, що поле по адреси (URL) та поле тіла запиту повинні приймати символи, як цифри так і літери. Адресний рядок вказує на адресу, за якою буде відправляти запит, а тіло запиту - передає додаткові данні, які потрібні для запитів.

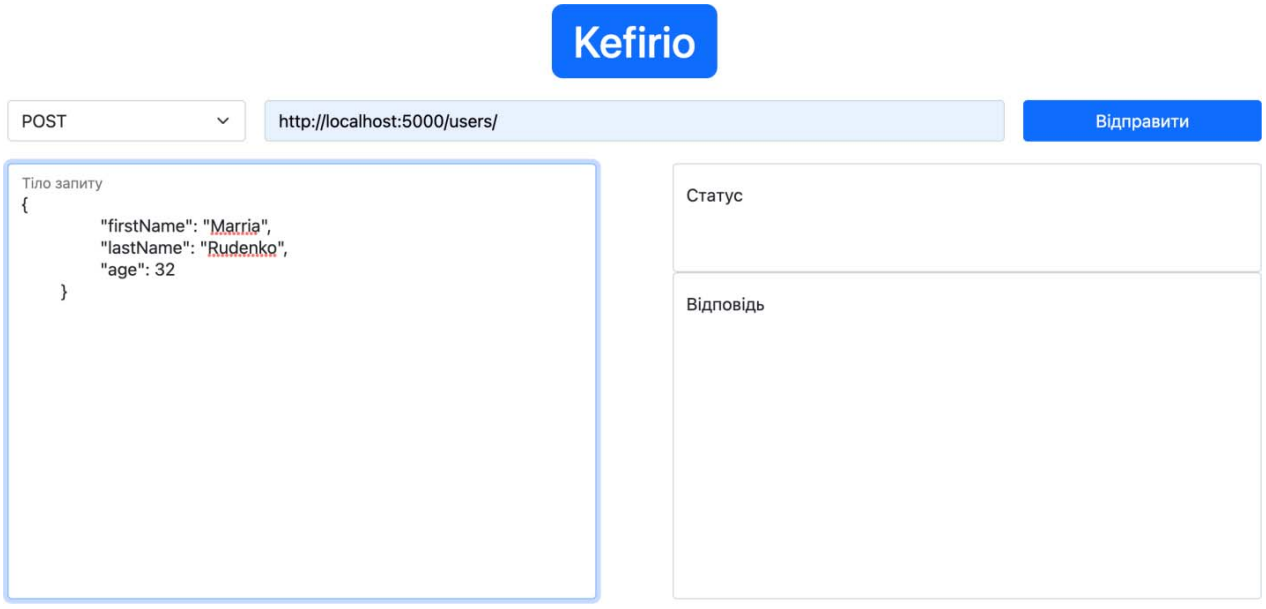


Рис..3.13. Введення в адресний рядок та тіло запиту

Основним завданням додатку є виконання запитів та отримання інформації про запити для тестування backend частини [23]. Тому наступним кроком є перевірка роботоспроможності самих запитів.

Першим необхідно перевірити можливість отримання інформації, для цього необхідно вибрати метод GET, вказати адресу ,за якою буде відправлятись запит та натиснути кнопку «Відправити». Очікуєть, що після виконання цих дій, в полі статус буде відображено статус виконання запиту та тіло відповіді в полі «Відповідь». Результат тесту відображений на рисунку 3.14.

Рис..3.14. Результат тестування запиту отримання інформації

Як видно з рисунку, тест є успішним і виконується як і очікувалось.

Наступним необхідно перевірити якість виконання методу POST та запису даних на backend. Для цього необхідно виконати схожі дії як і в минулому тесті, лише змінити тип запиту та додати тіло запиту (об’єкт з даними про користувача).Після вибору типу запиту, внесення даних в тіло запиту, та заповниши адресний рядок, необхідно натиснути кнопку

«Відправити». Очікуваним результатом є : модальне вікно про успішне створення нового користувача та при запиті отримання інформації - відображення ще одного користувача. Результат тесту відображено на рисунках

3.15 та 3.16.

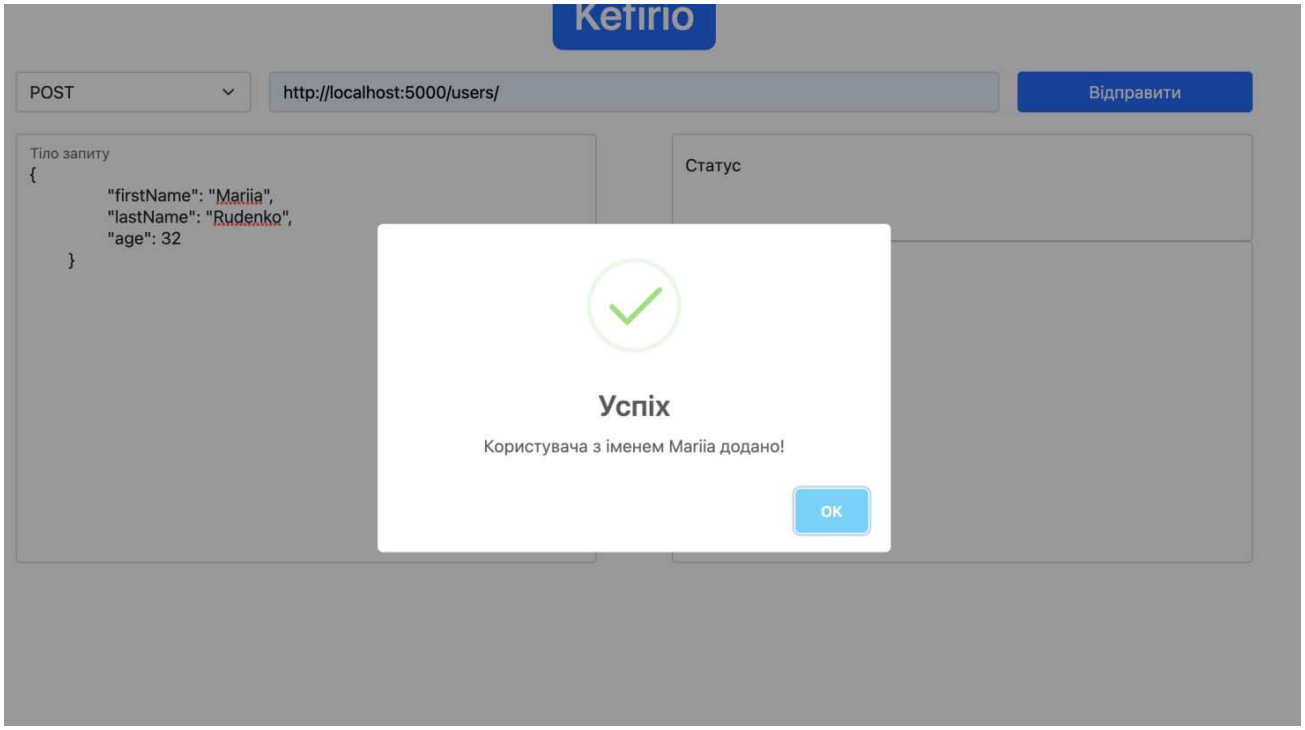


Рис..3.15.- Модальне вікно успішного виконання запиту

Рис..3.16. Відображення успішно створеного користувача при отриманніданих

Наступним запитом є запит заміни певних даних по користувача. Для виконання цього запиту необхідно змінити тип запиту на PATCH [24], вибрати

конкретного користувача та додати значення поля id в адресний рядок. Після цього в тіло запиту необхідно вказати поле, яке необхідно змінити та його значення (рис 3.17). Очікуваним результатом є поле, яке вказано в тілі запиту змінилось, інші поля цього об’єкту залишились незмінні та модальне вікно, про успішну зміну даних (рис. 3.18 та рис. 3.19).

Рис.3.17. Дані для зміни інформації про користувача

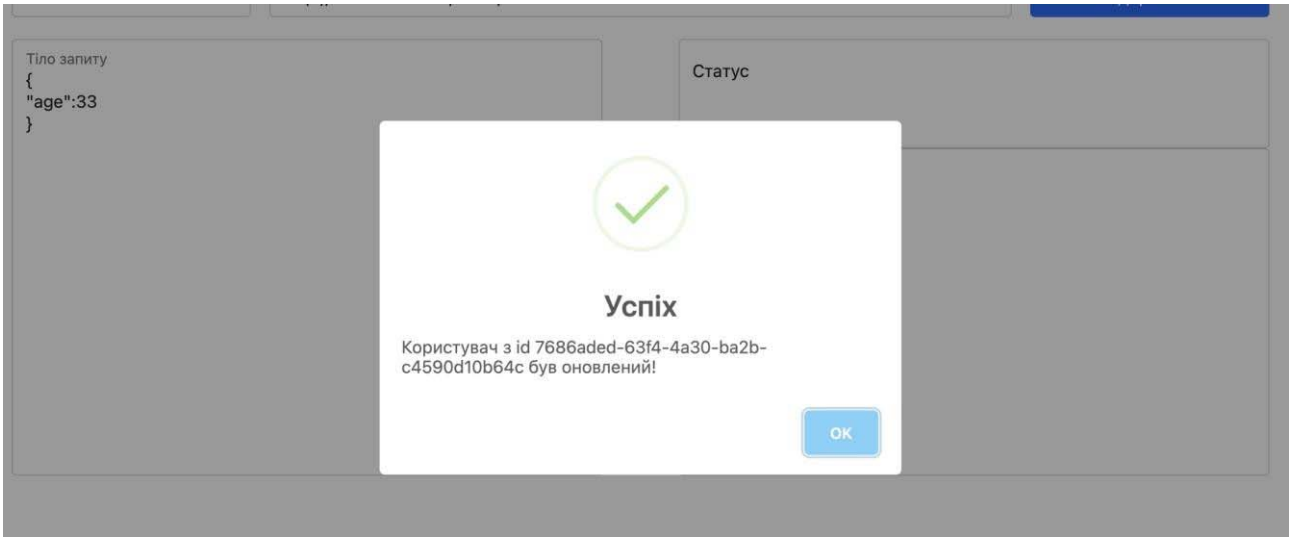


Рис.3.18. Модальне вікно про успішну зміну даних для користувача

Рис..3.19. Результат виконання методу зміни даних користувача

Як видно з рисунку 3.20, значення поля age змінилось з 32 на 33, при цьому всі інші значення інших полів - залишились незмінними.

Останньою операцією є операція видалення користувача. Для видалення користувача необхідно вибрати тип запиту DELETE, в адресному рядку крім адреси вказати значення id для цього користувача та натиснути на кнопку

«Відправити» (рис. 3.21). Очікується відображення модального вікна про успішне видалення (рис. 3.22) та при повторному запиті на отримання інформації про користувачів - даного користувача не буде в списку користувачів(рис. 3.23).

Рис. 3.20. Дані для видалення користувача

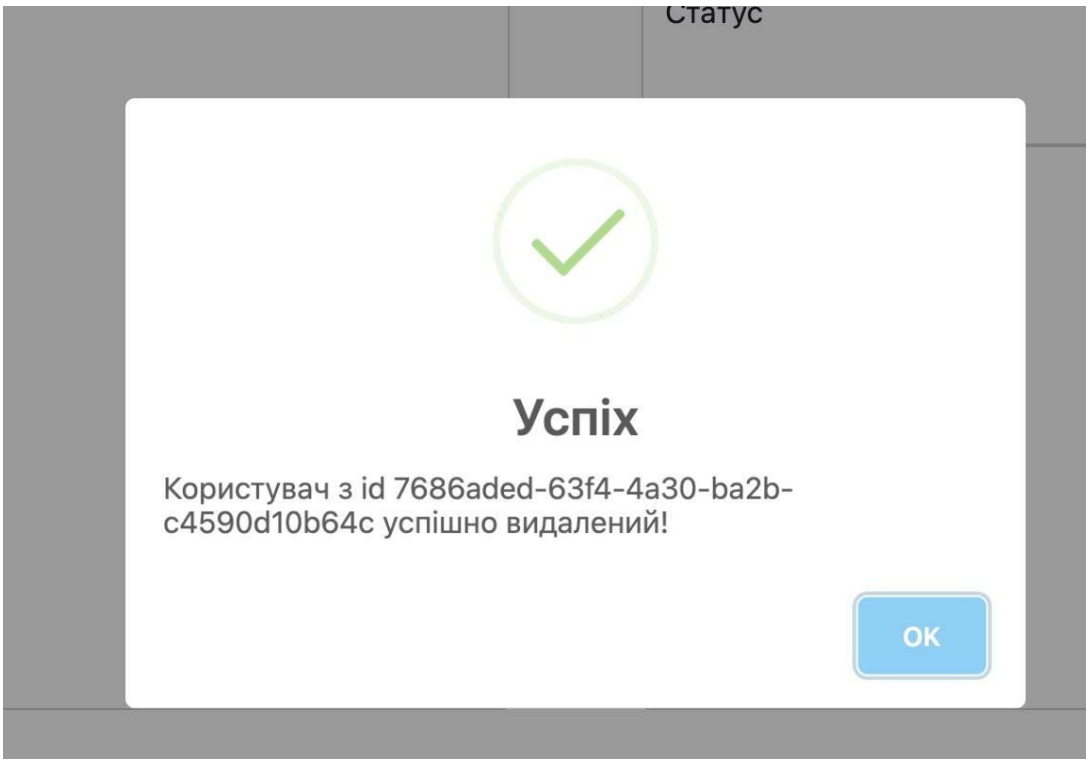


Рис. 3.21. Модальне вікно про успішне видалення користувача

Рис. 3.22. Результат отримання даних про користувачів після видалення

Як видно з результатів тестування, користувача було видалено та при повторному отриманні даних про користувачів в системі - видалення конкретного користувача ніяк не вплинуло на інших користувачів.

Висновки до розділу ІІІ

У цьому розділі кваліфікаційної роботи проведено огляд та аналіз технологій і засобів розробки програмних засобів для реалізації методів тестування backend частини додатків. В результаті аналізу було прийнято рішення, розробити програмний засіб у вигляді односторінковоговеб-додатку за монолітною архітектурою, де взаємодія між клієнтом та сервером забезпечується за допомогою REST API. Для реалізації програмного засобу було вибрано мову програмування JavaScript та середовище Visual Studio Code. Для розробки серверної частини було вибрано фреймворк Node.js, а для взаємодії з бекенд частинами додатків, бібліотке axios. Також було розроблено загальний алгоритм додатку. Розроблено додаток, в якому реалізовано метод отримання та запис данних, реалізовано метод видалення та зміни данних для отримання відповіді від сервера та його тестування.

Розглянуто та проаналізовано методи тестування. В результаті порівняння, було вибрано метод тестування чорного ящика для тестування програмного засобу. У результаті тестування, було підтверджено правильність роботи методів взаємодії з сервером та коректності роботи для тестування. Тестування програми показало повну працездатність і відповідність технічному завданню, що було поставлено.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

В кваліфікаційній роботі розглянуто аналіз стану програмних засобів для тестування, що показав що створення власного ПЗ є актуальним завданням. Було розглянуто такі аналоги як: Postman, SoapUI, JMeter. Порівняння уже створених аналогів показало, що є необхідність розробити власний додаток. Також було проведено аналіз методів реалізації програмного продукту. Було обрано методи роботи з даними. Було розглянуто методи Get, Post, Put, Delete,Patch. Після порівняння було обрано методи Get, Post, Patch, Delete. В результаті було розроблено основні завдання, які необхідно виконати при розробці програмного додатку.

Було проаналізовано методи та принципи тестування веб-додатків та було проаналізовано принципи тестування backend частини додатків, принцип обміну даних та основних потреб при тестуванні будь-якого додатку. Сформовано основні вимоги до методу та програмного засобу. Розроблено метод для реалізації програмного засобу для тестування backend частини за допомогою API через клієнтський інтерфейс.

Проаналізовано принципи створення інтерфейсів, внаслідок чого, було визначено основні потреби користувача і розроблено прототип майбутнього додатку. На основі прототипу розроблено дизайн-макет тобто інтерфейс програмного засобу.

Удосконалено метод тестування програмного коду backend частини веб-додатків, який на відміну від наявних, забезпечує тестера візуальним інтерфейсом, що дає змогу візуально оцінити важливість тестування кожної програмної функції, і в такий спосіб підвищити ефективність процесу тестування. Суть методу полягає в зміні стратегії тестування backend частини додатків, замінивши тестування API через програмний код на тестування API за допомогою візуального інтерфейсу, що виконує роль «мосту» між клієнтом і сервером, який в свою чергу дає можливість людям, які не знають певної мови

програмування, забезпечувати якість програмних продуктів за допомогою тестування API.

Проведено огляд та аналіз технологій і засобів розробки програмних засобів для реалізації методів тестування backend частини додатків. В результаті аналізу було прийнято рішення, розробити програмний засіб у вигляді односторінкового веб-додатку за монолітною архітектурою, де взаємодія між клієнтом та сервером забезпечується за допомогою REST API. Для реалізації програмного засобу було вибрано мову програмування JavaScript та середовище Visual Studio Code. Для розробки серверної частини було вибрано фреймворк Node.js, а для взаємодії з бекенд частинами додатків, бібліотке axios. Також було розроблено загальний алгоритм додатку. Розроблено додаток, в якому реалізовано метод отримання та запис данних, реалізовано метод видалення та зміни данних для отримання відповіді від сервера та його тестування.

Було розглянуто та проаналізовано методи тестування. В результаті порівняння, було вибрано метод тестування чорного ящика для тестування програмного засобу. У результаті тестування, було підтверджено правильність роботи методів взаємодії з сервером та коректності роботи для тестування. Тестування програми показало повну працездатність і відповідність технічному завданню, що було поставлено.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Регресія та обслуговування. URL: https://www.guru99.com/software-testing-introduction-importance.html#1
2. Postman. URL: https://medium.com/effective-developers/postman
3. JMeter. URL: https://habr.com/ru/post/418313/
4. Протокол HTTP. URL: https://qastart.by/class-2/21-metody-http-zaprosa
5. Автотестування сценаріїв. URL: https://unetway.com/tutorial/testing-software-types
6. Рівні сценаріїв. URL: https://unetway.com/tutorial/testing-software- methods
7. Тестування переносимості. URL: https://unetway.com/tutorial/testing-software-levels
8. Тестування API. URL: http://33testers.blogspot.com/2015/07/api.html
9. User story. URL: https://habr.com/ru/post/568360/
10. Інтерфейси користувача. URL: https://habr.com/ru/post/208966/