

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ТЕСТУВАННЯ API

DEVELOPMENT OF API TESTING SOFTWARE

Виконав: студент(ка) 4 курсу, групи КНТ-218

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

Шовак К.Г.

(прізвище та ініціали)

Керівник Гладкова О.М.

(прізвище та ініціали)

Рецензент Зеленьова І.Я.

(прізвище та ініціали)

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

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

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
“ ” 2022 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Шовака Кирила Георгійовича
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка програмного забезпечення тестування API.
Development of API Testing Software

керівник проєкту (роботи) Гладкова Ольга Миколаївна, к.т.н.,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “27” квітня 2022 року № 102

2. Строк подання студентом проєкту (роботи) 07 червня 2022 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне
завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Вибір інструментальних засобів розробки. 3. Розробка основних рішень програмного застосунку. 4. Керівництво з використання програмного продукту.

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

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Гладкова О.М., доцент		
Нормоконтроль	Липовець М.В., асистент		

7. Дата видачі завдання “04” квітня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Дослідження особливостей розробки системи. Вибір інструментарію розробки.	2-3 тижні	Розділ 2
4	Проектування системи.	4-5 тижні	Розділ 3
5	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Розділ 4, Додатки
6	Нормоконтроль та рецензування.	7 тиждень	
7	Захист роботи.	8 тиждень	

Студент(ка)

(підпис) Шовак К.Г.
(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис) Гладкова О.М.
(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 64 с., 24 рис., 4 табл., 2 дод., 15 джерел.

АВТОМАТИЗАЦІЯ ТЕСТУВАННЯ, СЕРВЕР, API, REST-ЗАПИТИ, POSTMAN.

Об'єкт дослідження – автоматизація тестування.

Предмет дослідження – програмні засоби для автоматизації тестування серверної частини проєкту.

Мета роботи – розробити забезпечення для автоматизації тестування серверної частини проєкту, тестування API .

Для вирішення поставленої мети необхідно вирішити наступні завдання:

- аналіз можливостей тестування API;
- вибір засобів розробки
- розробити структуру проєкту;
- розробити програмне забезпечення.

Матеріали, методи та технічні засоби: мова програмування JavaScript, технологія REST API, середовище Postman – HTTP клієнт для тестування API.

Результати. Розроблено програмне забезпечення для автоматизації тестування API, що дозволяє прискорити процес тестування серверної частини продукту.

Галузь використання – тестування програмного забезпечення.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 64 pages, 24 figures, 4 tables, 2 appendixes, 15 sources.

TESTING AUTOMATION, SERVER, API, REST-REQUESTS, POSTMAN.

The object of research is the automation of testing.

The subject of research - software to automate the testing of the server part of the project.

The purpose of the work - to develop software to automate the testing of the server part of the product, API testing.

To solve this goal it is necessary to solve the following tasks:

- analysis of API testing capabilities;
- choice of development tools;
- develop the project structure;
- develop software.

Materials, methods and technical means: JavaScript programming language, REST API technology, Postman environment - HTTP client for API testing.

Results. Developed software to automate API testing, which speeds up the testing process of the server part of the product.

Field of use – software testing.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок.....	7
Вступ.....	8
1 Аналіз предметної області.....	10
1.1 Дослідження процесу автоматизації тестування	10
1.2 Особливості тестування застосунків з клієнт серверною архітектурою.....	13
1.3 Висновки за розділом	17
2 Вибір інструментарію розробки систем	18
2.1 Огляд сторони серверу та методів тестування API	18
2.2 Вибір середовища для автоматизації тестування API.....	21
2.3 Висновки за розділом 2	26
3 Проєктування проєкту	28
3.1 Проєктування загальної архітектури проєкту.....	28
3.2 Проєктування структури програмного проєкту.....	30
3.3 Розробка запитів до серверу	31
3.4 Висновки до розділу 3	32
4 Керівництво з використання програмного продукту	33
4.1 Призначення програмного продукту	33
4.2 Вхідні та вихідні дані програмного продукту.....	33
4.3 Використання програмного продукту	34
4.4 Висновки до розділу 4	37
Висновки	38
Перелік джерел посилання	40
Додаток А Текст програми.....	42
Додаток Б Слайди презентації	57

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
JSON	– JavaScript Object Notation;
HTTP	– HyperText Transfer Protocol;
RDF	– Resource Description Framework;
REST	– Representational State Transfer;
SOAP	– Simple Object Access Protocol;
URI	– Uniform Resource Identifier;
URL	– Uniform Resource Link;
URN	– Uniform Resource Name;
БД	– база даних.

ВСТУП

Роботу застосунку який має клієнтську і серверну частину з базою даних, загалом можна описати як: викликати команди на сервері та отримувати від нього відповіді згідно з протоколом, що описує взаємодію клієнтської та серверної частин. Ці дві частини робляться різними розробниками, і, зазвичай, клієнтська частина робиться вже по тому, як серверна буде готова.

Тестування частини програми, розміщеної на вебсервері, можна провести і обминаючи графічний (клієнтський) інтерфейс, проте це вимагає від спеціаліста певного рівня знань та навичок технічного характеру, а також застосування додаткових інструментів. Розглянемо вебсервер з погляду навантажувального та інсталяційного тестування.

Те, що складові вебзастосування взаємодіють між собою за допомогою HTTP (HyperText Transfer Protocol), є гарною новиною для тестування, так як це дозволяє не просто відстежувати спілкування, вникаючи в логіку роботи і звіряючи її з технічними вимогами, а й впливати безпосередньо на застосунок, прикладаючи свою руку до запитів, що відправляються.

Послідовні цикли розробки, особливо у великих компаніях вимагатимуть багаторазового виконання одного й того ж набору тестів. Використовуючи інструмент автоматизації тестування, можна записати цей набір тестів і за необхідності відтворити його. Після автоматизації набору тестів втручання людини не потрібне. Це покращило процес тестування. Мета автоматизації – зменшити кількість тестових прикладів, які потрібно запускати вручну, а не виключити ручне тестування.

Ручне тестування всіх робочих процесів, полів та негативних сценаріїв вимагає більше часу та грошей (за певних умов). Не потребує втручання людини. Можна запустити та перейти до інших завдань. Збільшує швидкість виконання тестів. Допомагає збільшити тестування. Ручне

тестування може набриднути, і наслідками стануть втрата уваги та поява помилок.

Автоматизація тестових сценаріїв це хороший спосіб підвищити ефективність, а також збільшити охоплення та швидкість тестування програмного забезпечення, коли вам потрібно повторювати одні й ті самі тестові сценарії.

Таким чином, розробка програмного забезпечення для автоматизації тестування API (Application Programming Interface), тобто тестування надсилання даних до серверу та отриманих відповіді або даних результатів від сервісу є актуальним завданням.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Дослідження процесу автоматизації тестування

Процес автоматичного тестування представляє собою процес автоматизованого тестування, від вибору інструменту тестування до виконання тестового сценарію. Як показав аналіз роботи [1] процес автоматизованого тестування складається з декількох кроків які можна представити у вигляді алгоритму, який зображений на рисунку 1.1.



Рисунок 1.1 – Алгоритм автоматизованого тестування

Крок 1. Вибір інструменту тестування.

Для автоматизації тестування необхідно вибрати те, що найкраще відповідає потребам проєкту. Наприклад, якщо мета полягає в тому, щоб виявити певну помилку в програмному забезпеченні, можна використовувати тестування автоматизації аналізу коду.

Можна вибрати з широкого спектру інструментів автоматизації тестування та вебзастосунків на ринку, таких як Selenium IDE, WebDriver, UFT, Ranorex, Cucumber, Testcomplete і Appium. Тож потрібно мати чітке розуміння кожного інструменту та того, як він може бути корисним для тестування.

Крок 2. Встановити сферу автоматизації.

Це стосується тесту області, яка буде автоматизована, і того, наскільки великим вона буде. На цьому кроці необхідно ознайомитися зі станом команди, підготувати дані тесту та середовище, в якому має проходити тест. Автоматичне тестування означає, що втручання людини рідко потрібне, тому можна залишити тест для виконання без уваги людини.

Ось деякі додаткові фактори, які слід враховувати при визначенні обсягу автоматизованого тестування: складність тесту, основні цілі тестування, скільки ресурсів і бізнес-компонентів використовується, технічна можливість.

Крок 3. Етап планування.

На етапі планування розглядається створення стратегії тестування. Це включає в себе інструмент тестування автоматизації та те, як збирається його використовувати, дизайн і функції, а також цілі. Плануються те що необхідно дізнатися в кінці тестування. Також потрібно буде створити детальний графік написання сценаріїв і виконання тестових випадків разом з командою розробників, яка буде проводити тести. Також розглянути елементи автоматизації в межах і поза межами документації.

Крок 4. Етап виконання.

Це передостанній етап у процесі автоматизованого тестування. Коли є інструмент і стратегія тестування, настав час запуснути тест. Залежно від вибраного тесту, розробникам може знадобитися написати код, вивчити мову опису вебслужб і запуснути тест автоматично з самого початку за допомогою API інструменту або інтерфейсу користувача.

Перед початком роботи також може знадобитися тестування API. Автоматизовані тести також створять звіт, який можна проаналізувати разом із рештою команди. У ньому подано підсумок тестування на даний момент. Це можна використовувати в майбутніх тестах для порівняння.

Крок 5. Поточне обслуговування.

Поточне обслуговування має бути частиною процесу автоматизованого тестування. Це особливо необхідно, якщо планується в майбутньому запускати тести з багаторазовими тестовими сценаріями. Оскільки, навіть якщо буде готовий до використання сценарій, його все одно потрібно буде оновити до моменту виконання наступного тесту, якщо інструмент змінився.

Постійне технічне обслуговування також дає заспокоєння, коли команда пробирається до наступного етапу або повертається, щоб провести ще один тест. Якщо метод був неодноразово випробуваний, швидше за все, буде отримано точний результат. Необхідно знайти потрібні інструменти, щоб зробити автоматизацію легкою.

Автоматизація може справді змінити спосіб ведення бізнесу. Зараз настав час використовувати нові технології та вивчати методи, щоб зробити робочий день ефективнішим. Інколи треба спробувати кілька різних інструментів, щоб побачити, який з них найкраще підходить.

Використання автоматизованого тестування – це найкращий спосіб упевнитися, що завжди в курсі налагодження, дефектів і проблем, які можуть швидко погіршити виробничий процес, якщо їх не вирішити якомога швидше [1].

1.2 Особливості тестування застосунків з клієнт серверною архітектурою

На рисунку 1.2 зображена стандартна клієнт-серверна архітектура на базі якої будуються всі застосунки з даною архітектурою.



Рисунок 1.2 - Архітектура клієнт-сервер

Архітектура «клієнт-сервер» визначає загальні принципи організації взаємодії в мережі, де є сервери, вузли-постачальники деяких специфічних функцій (сервісів) та клієнти (споживачі цих функцій).

Основні моделі взаємодії у рамках дволанкової архітектури [2]:

- сервер терміналів – розподілене подання даних;
- файл-сервер — доступ до віддаленої бази даних та файлового файлу ресурсів;
- сервер БД — видалене представлення даних;
- сервер програм — віддалена програма.

Компоненти у триланковій архітектурі розподіляються наступним чином:

- подання даних — на стороні клієнта;
- прикладний компонент — на виділеному сервері програм;
- управління ресурсами - на сервері БД, який і представляє запитувані дані.

Все програмне забезпечення для роботи з протоколом HTTP в багатоланковій архітектурі поділяється на три великі категорії:

- сервери як основні постачальники послуг зберігання та обробки інформації (обробка запитів);
- клієнти – кінцеві споживачі послуг сервера (надсилання запиту);
- проксі для виконання транспортних служб.

Проксі-сервер (проху - "представник, уповноважений") проміжний сервер (комплекс програм) в комп'ютерних мережах, що виконує роль посередника між користувачем та цільовим сервером, що дозволяє клієнтам як виконувати непрямі запити до інших мережевих служб, так і отримувати відповіді [2].

Як правило, клієнт – це браузер, але трапляються і винятки (у тих випадках, коли один вебсервер виконує запит до іншого, роль клієнта відіграє вебсервер один. У класичній ситуації (коли роль клієнта виконує браузер) для того, щоб користувач побачив графічний інтерфейс програми у вікні браузера, останній повинен обробити отриману відповідь вебсервера, в якому буде міститися інформація, реалізована із застосуванням HTML, CSS, JS). Саме ці технології «дають зрозуміти» браузеру, як саме необхідно «відмалювати» все, що він отримав у відповіді.

Вебсервер – сервер, який приймає HTTP-запити від клієнтів та видає їм HTTP-відповіді. Щоб уникнути можливої плутанини, вебсервером називають як програмне забезпечення, яке виконує функції вебсервера, так і безпосередньо комп'ютер, на якому це програмне забезпечення працює [3].

Для однозначної ідентифікації ресурсів у мережі Інтернет використовуються унікальні ідентифікатори URI (Uniform Resource Identifier).

Єдиний ідентифікатор ресурсу URI є короткою послідовністю символів, що ідентифікує абстрактний або фізичний ресурс. URI не вказує на те, як отримати ресурс, лише ідентифікує його. Це дозволяє описувати за допомогою RDF (Resource Description Framework) ресурси, які не можуть бути отримані через Інтернет (імена, назви тощо). Найвідоміші приклади URI - це URL (Uniform Resource Link) та URN (Uniform Resource Name) [4].

Отже, перш ніж приступити до тестування, необхідно встановити (інсталювати) вебпрограму на вебсервер. П'ять особливостей тестування послідовно перераховані нижче [5].

Більшість вебзастосунків потребує встановлення специфічних знань в адмініструванні операційної системи. Спробувати встановити програму на кількох вебсерверах. В оптимальному випадку це будуть найпопулярніші технології серед користувачів, для встановлення списку яких потрібно попереднє дослідження.

Інсталюючи вебзастосунок, звертати увагу на те, чи дійсно програма встановлюється у вказану директорію, базу даних, використовує вибраний префікс та дотримується інших конфігураційних моментів.

Виконувати негативні тести: перевірки в умовах недостатності ресурсів (місця на диску, оперативної пам'яті) або привілеїв, переривання процесу встановлення під час активної його фази (наприклад, надсилаючи сервер у перезавантаження).

Навантажувальне тестування імітує роботу з програмою певної кількості користувачів. Цей вид тестування здійснюється за допомогою спеціальних інструментів (наприклад, jMeter), головна мета яких – визначити профілі навантаження та штучно створити для них навантаження, що виявляє граничні можливості програми (або сервера) в умовах роботи з ним тієї чи іншої кількості користувачів.

При цьому обов'язково враховувати [5]:

- конфігурацією сервера і показники ресурсів- обсяг оперативної пам'яті, жорстких дисків, число ядер тощо;
- розмір бази даних при великому заповненні фіктивними або реальними даними - великою кількістю користувачів, замовлень, повідомлень тощо;
- облік нерівномірного навантаження на специфічні сторінки або час;
- облік запущених фонових процесів сервера.

Отримана інформація піддається ретельному аналізу з подальшим виявленням «вузьких місць» та граничних програмних та апаратних можливостей, які надалі використовуються для забезпечення стабільності вебсервера та самого застосунка, що працює на ньому.

Ще однією раніше розглянутою складовою вебзастосунків є база даних, де застосунок зберігає всю необхідну інформацію.

Для того, щоб база даних слугувала гідним сховищем інформації для вашої програми, при тестуванні необхідно звертати увагу на такі основні моменти [5]:

- інформація, що вноситься через інтерфейс, повинна бути збережена в базі даних у незмінному (початковому) вигляді;
- збережена в базі даних інформація повинна відображатися в будь-якій частині програми однаково (якщо іншого не вимагає бізнес-логіка програми);
- назви таблиць та структура бази даних (БД) повинні відповідати проєктній документації;
- потрібно стежити за тим, щоб запити не оброблялися надто довго, а кількість з'єднань була достатньою, моніторинг стану БД – один із важливих моментів тестування;
- варто враховувати, що видалення запису в БД який завжди супроводжується повним видаленням сутності, іноді використовується так зване псевдовидалення, і потрібно перевірити, чи правильно воно виконується;
- порожню БД на тестовому стенді рекомендується заповнювати згенерований випадковими даними.

Усі складові вебпрограми повинні взаємодіяти між собою, і це відбувається завдяки HTTP(s). Без HTTP багатостороння система не функціонувала б у принципі, тому що HTTP – це протокол передачі даних, який займає одне з основних місць у нашій клієнт-серверній архітектурі [6].

Взаємодія здійснюється через повідомлення (запити та відповіді): на надісланий запит від клієнта має прийти відповідь сервера. Зазвичай запит/відповідь складається з декількох складових, вони зображені на рисунку 1.3.

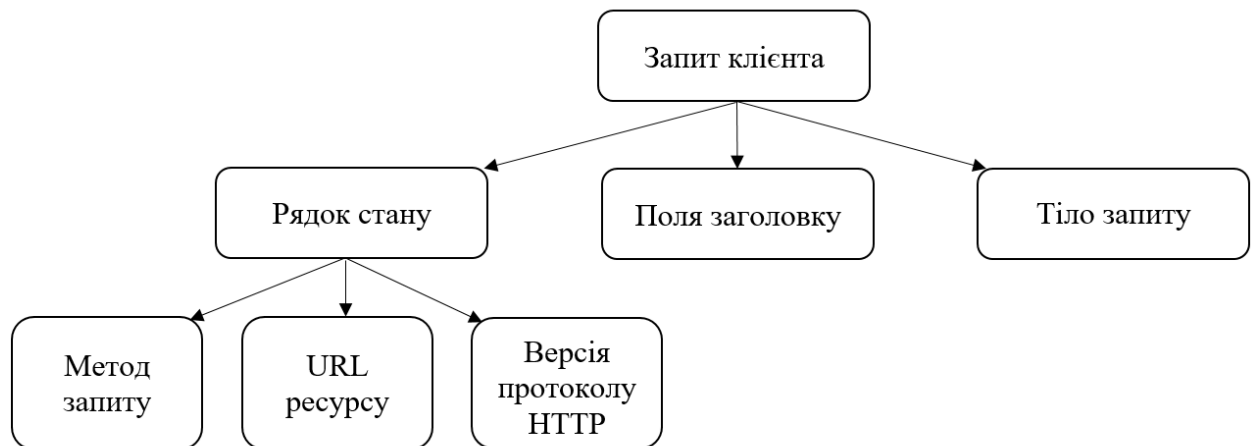


Рисунок 1.3 – Загальна структура запиту

При роботі з відповідями фахівець із тестування насамперед повинен звертати увагу на методи та коди стану, які присутні у стартовому рядку (рядок стану).

1.3 Висновки за розділом

В розділі було проаналізовано процесу автоматизації тестування. Особливістю тестування клієнт-серверних застосунків є наявність декількох частин проєкту для тестування. Тестування сторони серверу є одним з перших, після чого йде тестування клієнтської частини. Тестування серверної частини дозволяє виявити помилки на ранній стадії. І чим раніше будуть виявлені помилки на стороні серверу тим швидше виконається подальша розробка та тестування клієнтської частини. Як показало дослідження з цього розділу основним способом тестування серверу є тестування взаємодії через HTTP запитів. Загальна структура запитів та відповідей між різними частинами клієнта та сервера наведена у розділі.

2 ВИБІР ІНСТРУМЕНТАРІЮ РОЗРОБКИ СИСТЕМ

2.1 Огляд сторони серверу та методів тестування API

Як показує досвід роботи в сфері тестування, зазвичай проєкт складається з сторони сервера – вебсервіси, та сторони клієнта. Найчастіше серверна сторона виконується першою, а клієнтська завершується пізніше. Таким чином тестування серверної сторони є першим кроком тестування проєкту.

Вебсервіси та їхні різні протоколи — це перше, чого повинні засвоїти розробники автоматизації, намагаючись спеціалізуватися на тестуванні на стороні сервера. Існують два найпопулярніші вебсервіси.

Перший це Simple Object Access Protocol (SOAP), або простий протокол доступу до об'єктів – протокол для обміну повідомленнями. SOAP є попередником наступного сервісу REST (Representational State Transfer) [7].

Другий REST – передача представницького стану, REST є популярною технологією побудови взаємодії між клієнтом і сервером [8].

Структури даних – це типи даних, які ми передаємо між службами, це різні формати, які в кінцевому підсумку представлені файлами, більшість із яких у форматах XML та (в першу чергу) JSON.

Основна частина тестування на стороні сервера здійснюється у формі тестування API – запити API надсилаються іншій стороні, очікуючи, що вона поверне відповідь (яку пізніше можна аналізувати, компілювати та тестувати). Немає графічного інтерфейсу, елементів чи об'єктів (які існують у клієнтському тестуванні, наприклад: кнопки, текстові поля тощо).

Тут необхідно працювати з інструментами та фреймворками тестування API, такими як: REST Assured , REST Sharp, Http Client. Важливим моментом тут є знання HTTP запитів [7].

Складові проєкту мають взаємодіяти один з одним, і це відбувається з використанням HTTP(s). Взаємодія виконується за допомогою повідомлення

(запити й відповіді): на відправлений запит від клієнта надходить відповідь сервера. Класичний запит/відповідь складається з 3 складових (рис 1.3):

- стартовий рядок;
- заголовок;
- тіло повідомлення.

Найпопулярнішими методами які використовуються в стартовому рядку є GET, HEAD, POST та інші, опис наведений в таблиці 1.1 [8].

Таблиця 1.1 – Призначення методів GET, POST, HEAD та інші

Метод	Призначення
Метод GET	Використовується для запиту вмісту на сервері (наприклад, GET /path/resource?param1=value1¶m2=value2 HTTP/1.1).
Метод POST	Даний метод використовується для передачі даних на сервер, проте його основа "ховається" в тіло, що відрізняє його від GET.
Метод HEAD	За своєю суттю не відрізняється від вищезазначеного методу, проте відповідь сервера на такий запит позбавлений «тіла», а практичне застосування орієнтоване на полегшене використання з метою отримання мінімальної інформації про сервер/продукт або його статус.
Метод PUT	Даний метод використовується для заміни одних значень іншими. На відміну від POST він повністю видаляє попереднє значення і встановлює нове, POST в сою чергу дозаписує до вже існуючого значення нове.
Метод DELETE	Видаляє на сервері дані запиту.

Існують також інші методи: CONNECT, TRACE, PATCH тощо.

Класичними програмами, які можна використовувати для створення запитів, є Fiddler або Postman. Використовуючи Fiddler, можна легко відстежувати всі запити від клієнта та відповіді, переглядати їх деталі, а також вносити свої зміни та відправляти модифіковані запити на сервер, оцінюючи поведінку системи в такому випадку.

В запиті необхідно звертати увагу на нижче перераховані моменти.

Чи правильний метод використовується для того чи іншого запиту. Якщо відправляється повідомлення, а на сервер йде лише GET-запит, то тут щось явно не так.

Здавалося б, клік по одній і тій же функціональній клавіші генерує один і той самий запит. Але є випадки, коли клік по кнопці приводив до ситуації, в якій JavaScript переписував запит кнопки, що знаходиться поруч, таким чином змінюючи її призначення.

Вникати у запити, що надсилаються. В них досить легко розібратися, особливо якщо це GET – вони логічно зрозумілі навіть пересічному користувачеві. Аналіз запитів - це можливість виявити дефект, що сховався, набагато швидше, ніж здійснюючи його пошук в інтерфейсі.

Наглядати за трафіком на запити на інші сервери. Приклад із життя: фронтенд сайту робив фріланс-розробник, після завершення роботи якого необхідно було тестування. Мати звичку моніторити весь трафік програми, що тестується: це дозволяє виявити, що вищезгаданий розробник без зазріння совісті сховав у «фронт» сайту запити, які працювали на благо його особистого інтернет-магазину.

Монітору трафік, уважно слідкуйте за кодами станів.

Кожна відповідь на будь-який запит несе в собі багато корисної для розробників та фахівців з тестування програмного забезпечення інформації. Одним із джерел такої інформації може стати код стану: за цим кодом клієнт дізнається про результати його запиту та визначає, які дії йому вжити далі. Набір кодів стану є стандартом, описаний у відповідних документах RFC. Щоразу, коли створюється баг-репорт про непрацюючі посилання, інші елементи вебсторінки або системні помилки, обов'язково вказувати відповіді сервера: вони заощадять час розробнику при визначенні причин дефекту [7].

Усі коди можна розділити на групи (соті, двохсоті, трьохсоті, чотирьохсоті та п'ятисоті) кожна група-«сотня» несе свій тип інформації.

Групи кодів та тип інформації представлені в таблиці 1.2. Більш детально з кодами стану можна ознайомитись у [8].

Таблиця 1.2 – Огляд типів кодів відповідей від серверу

Код	Тип інформації	Призначення
1xx	Інформаційні	Інформування про передачу повідомлення. Самі повідомлення від серверу містять тільки початковий рядок відповіді, якщо потрібно, декілька специфічних для відповіді полів заголовку.
2xx	Повідомлення про успішну операцію	Інформування про успішну обробку запиту клієнта. Класичним і розповсюдженим є відповідь «200 OK»
3xx	Перенаправляючі	Повідомляють, що для успішного виконання операції необхідно зробити інший запит. Даного типу коди: 301, 302, 303, 305, 307. Адреса, за якою клієнту необхідно призвести запит, сервер вказує у заголовку Location.
4xx	Помилка на стороні клієнту	Вказує, що помилка на стороні клієнта, тобто в тілі запиту.
5xx	Помилка на стороні серверу	Вказує на помилку виконання операції за виною вебсервера.

На практиці, використовуючи при тестуванні спеціальні програми, в яких легко можна відсортувати запити і відповіді за кодом стану і відібрати, наприклад, всі 400-і і 500-і з подальшим їх аналізом. Таким чином дуже швидко «виловлюються» дефекти з «стилі, що відвалилися», скриптами, файлами, функціями програми і т.п.

2.2 Вибір середовища для автоматизації тестування API

Програмні інтерфейси застосунків (API) – це специфікація, яка працює як інтерфейс між компонентами програмного забезпечення [9].

Більшість функціонального тестування пов'язана на інтерфейсі користувача - формах, вебсторінках. API-тестування обходить інтерфейс

користувача, спілкуючись з застосунком безпосередньо через запити до його API.

Це тестування дозволяє тестувати такі технології, як JMS HTTP, бази даних і Вебслужби.

Іноді його називають «консольним» тестуванням. Більшість консольного тестування пов'язана на обхід інтерфейсу та відправлення запиту безпосередньо до бекенду або служби програми, а також отримання відповіді та посвідчення, що все працює так, як має бути.

Цей простий приклад часто описується як взаємини клієнта та сервера. Клієнт робить запит, звертаючись до ресурсу. Запит відправляється, знаходить сервер і сервер виконує запит. Він знаходить потрібний ресурс та відправляє відповідь клієнтської частини [9].

Нижче перераховані інструменти тестування API та наводиться зведене порівняння найпопулярніших у таблиці 1.3.

Таблиця 1.3 – Порівняння популярних інструментів для автоматизації тестування API

Характеристика	SoapUI	Postman	Katalon
Тестування REST API	+	+	+
Автоматична генерація інструкцій	+	+	+
Керування даними	JSON, CVS	JSON, CVS	Excel, JSON, CVS
Тестові звіти	HTML	JSON, HTML	HTML
Мова написання скриптів	Groovy	JS	Groovy, Java
Ціна	30 днів безкоштовно - 755\$ на рік	безкоштовно — 12 \$ /місяць	безкоштовно — 1899 \$ /рік

SoapUI - консольний інструмент, створений SmartBear. SoapUI це безголовий інструмент функціонального тестування, призначений для

тестування API, що дозволяє користувачам легко тестувати API REST і SOAP і вебсервіси [10].

Він існує у двох версіях – безкоштовна версія з відкритим вихідним кодом та Pro. Безкоштовну, отже, можна за необхідності модифікувати, а платна дружелюбніша до користувача і включає додатковий функціонал – редактор форм, майстер асертів для Xpath, редактор запитів SQL.

Безкоштовний пакет: Використовуючи безкоштовний пакет SOAPUI, користувачі можуть отримати повний вихідний код і створити свої бажані функції.

Швидко й легко створюйте тест за допомогою перетягування та клацання

Повторне використання сценаріїв: навантажувальні тести та сканування безпеки можна повторно використовувати для функціональних тестових випадків лише за кілька кроків [11].

Пакет Pro:

- потужне тестування на основі даних: дані завантажуються з файлів, баз даних і Excel, щоб вони могли імітувати, як споживачі взаємодіють з API;

- підтримка власної інтеграції CI/CD, асинхронне тестування.

Якщо команда займається лише тестуванням API і складається здебільшого з тестувальників, то це непоганий вибір. Це повнофункціональний інструмент тестування API, який не вимагає створення нового рішення з нуля. Якщо потрібна специфічна функціональність, її можна запрограмувати у SoapUI на Groovy. Інструмент добре підходить для складних сценаріїв тестування.

Можливості безкоштовної версії [11]:

- просте створення власного коду через Groovy;
- створення тестів методом перетягування;
- можливість створення складних сценаріїв;
- асинхронне тестування;

- mock-служба дозволяє імітувати поведінку служб до їхнього впровадження.

Платформа Katalon – це безкоштовне рішення для автоматизації тестування API, вебзастосунків, настільних і мобільних застосунків. Він швидко стає помітним інструментом для тестування API/вебсервісів, рекламуючи себе як повне наскрізне рішення для автоматизації для розробників і тестувальників [12].

Katalon Studio підтримує запити SOAP і REST, а також різноманітні команди та функції параметризації. Крім того, здатність Katalon Studio поєднувати інтерфейс користувача та API/вебсервіси для різних платформ (Windows, Mac OS і Linux) була названа відмінною рисою серед найкращих інструментів API.

Основні особливості [13]:

- підтримка як SOAP, так і Rest;
- універсальний магазин API, WebUI, настільного застосунка, мобільного тестування та комбінованих можливостей;
- підтримує підхід на основі даних;
- підтримка автоматизованого та дослідницького тестування;
- підтримка інтеграції CI/CD;
- підтримка AssertJ, однієї з найпотужніших бібліотек тверджень, щоб створити плавне твердження зі стилем BDD;
- підходить як для не технарів, так і для професіоналів у режимах ручного та Groovy Scripting.

Postman – це клієнт, який зародився як плагін для Chrome, але був випущений як десктоп-застосунок для Mac та Windows [14].

Багатьом не хочеться морочитися з програмуванням мовою розробників та середовищ розробки. Postman для них - ідеальний варіант для швидкого тестування API. При цьому він досить потужний і дозволяє створювати більш інтегровані рішення за бажання.

Postman має багатий інтерфейс, що полегшує його використання. Він також дозволяє легко ділитися знаннями з колегами – можна зберігати всі запити та очікувані відповіді та надсилати їх комусь ще.

Компанія Big Fish Games, що займається казуальними іграми, використовує Postman для спільної роботи команд. За словами Ембер Рейс, інженеру з тестування цієї компанії, хтось один відповідає за створення тест-кейсів для API. Ця людина потім публікує інформацію у wiki, і решта мають можливість запускати ці тести і переконатися, що зміни не ламають існуючий функціонал. Postman навіть дозволяє розміщувати на внутрішньому сайті кнопку "Запустити в Postman", що автоматично стартує тести [9].

Якщо команда хоче не тільки тестувати API, але й автоматизувати низку зусиль з дослідницького тестування, Postman – відмінний вибір.

Його можна використовувати для надсилання запитів POST до сервера та отримання відповідей. Він дозволяє налаштувати всі заголовки та тимчасові файли, які очікує ваш API, і перевірити відповідь на отримання.

Можна використовувати для автоматизованого та дослідницького тестування.

Можна використовувати для Mac, Windows, Linux & Chrome.

Широкі можливості для інтеграції – наприклад, із форматами Swagger & RAML.

Має функції запуску, тестування, документації та відстеження.

Не потребує вивчення нових мов.

Простий у використанні клієнт REST.

Багатий інтерфейс, що робить його простим у використанні.

Дозвольте користувачам легко ділитися знаннями з командою, оскільки вони можуть упакувати всі запити та очікувані відповіді, а потім надіслати своїм колегам [13].

Починаючи з версії 7.3, Postman допомагає користувачам краще впорядковувати колекції та елементи API (модельний сервер, монітори, тести

та документацію), генеровані зі схем API, за допомогою нових розширених налаштувань. Версія 8 з «Попередженнями» під час перевірки API, з додаванням попереджень безпеки, увімкнених для всіх API на Postman.

Плюси Postman [15]:

Інструмент, з яким знайомі багато хто, і він вкрай простий у користуванні.

Запити пишуться швидко, і для цього не потрібно жодного коду, а отже, ними зможуть скористатися всі без винятку.

Вкрай швидкий старт тестів.

Вбудована можливість ділитися колекцій усередині робочої команди для платних облікових записів.

Генерація документації за створеними запитам та збереженими відповідями.

Мінуси [15]:

При написанні параметрів запиту у формі form-data, як і раніше, зустрічаються прикрі неприємності, на кшталт неможливості передачі булевого значення. Вони просто наводяться до рядка, хоча числові значення працюють нормально.

Приблизно така ж проблема із вкладеними параметрами.

Обидва мінуси з легкістю можна обійти, якщо використовувати raw форму запису, але все ж таки неприємно. А ось плюси виглядають значно вражаюче.

2.3 Висновки за розділом 2

Було проаналізовано три найпопулярніші середовища автоматизації тестування API. З цього аналізу було обрано Postman, можна зробити висновок, що написання запитів у Postman проходить простіше, швидше і виглядає більш читабельним. Що стосується виконання самих запитів, то тут різниця не суттєва. Крім того мова JavaScript є простішою для вивчення и

використання. Безкоштовна безлімітна версія Postman є також значною перевагою оскільки функціоналу для початкового тестування більш ніж досить.

Також просто не можу не відзначити чудовий функціонал із генерації документації. Це необхідно, якщо ваш API ще не має нічого подібного. У ній ви можете описати весь API в цілому, надати приклади запитів, відповідей та опис окремих параметрів. Для цього можна вибрати одну з кількох запропонованих мов. Усе це щастя можна розмістити як публічно, і приватно.

3 ПРОЄКТУВАННЯ ПРОЄКТУ

3.1 Проєктування загальної архітектури проєкту

Першим за все на основі клієнт серверної архітектури було побудовано загальну архітектуру проєкту оминаючи графічний інтерфейс клієнтської частини за допомогою використання середовища Postman. Основою цієї архітектури є REST технологія. Розроблена загальна архітектура зображена на рисунку 3.1.

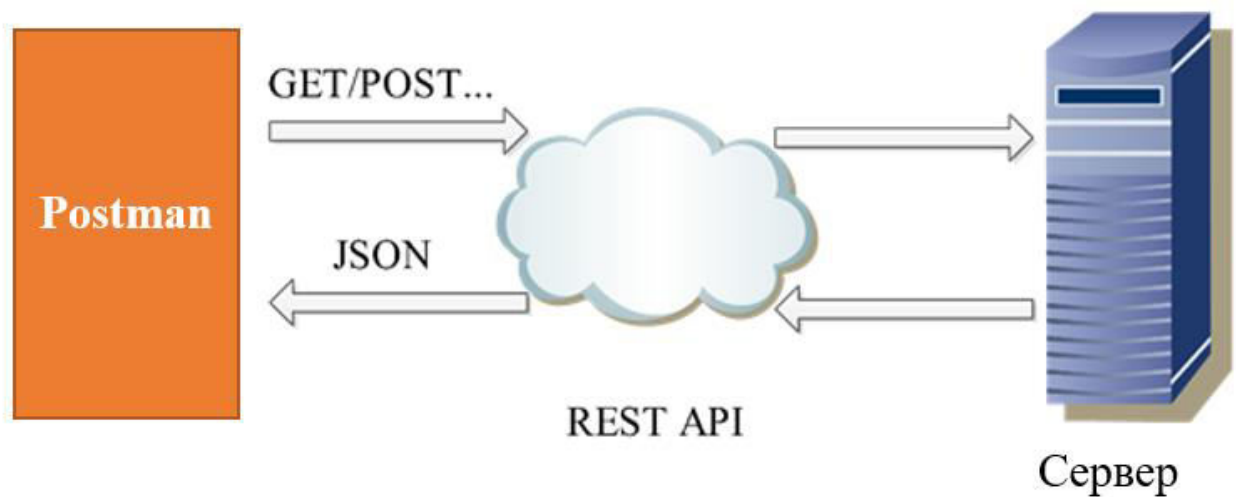


Рисунок 3.1 – Загальна архітектура проєкту

Сервер розроблюваного продукту надає REST інтерфейс для обміну даними. Цей сервер отримує запити та надсилає відповідь на ці запити. Отримані данні відповідей структурно належать архітектурі JSON, тобто представлено у форматі коду JavaScript.

Алгоритму роботи проєкту зображений у вигляді UML діаграми на рисунку 3.2 та включає чотири кроки які перераховані далі.

Крок 1. Відправка запиту з Postman. Замість клієнтської програми Postman відправляє запит структурований як HTTP метод (наприклад, GET) в інтерфейс вебAPI на стороні серверу.

Крок 2. API на стороні сервера отримує та обробляє запит. Налаштоване на стороні сервера API отримує, обробляє запит та взаємодіє з базою даних щоб отримати необхідну інформацію.



Рисунок 3.2 – UML-діаграма алгоритму роботи проєкту

Крок 3. API надсилає відповідь. Через Інтернет API надсилає відповідь на запит, у вигляді структурованого формату JSON.

Крок 4. Postman отримує й обробляє відповідь. Оскільки клієнтської програми немає і середовище виконує роль отримувача, тож Postman отримує відповідь у форматі JSON та надає інформацію користувачеві через програмний інтерфейс.

3.2 Проектування структури програмного проєкту

Наступним кроком є розробка структури програмного проєкту. На рисунку 3.3 зображена структура проєкту у вигляді структурованих директорій за логікою запитів. Проєк являє собою застосунок для реєстрації користувача, внесення банківських даних в систему, перегляд інформації на дашборді, та виконання грошових розрахунків.

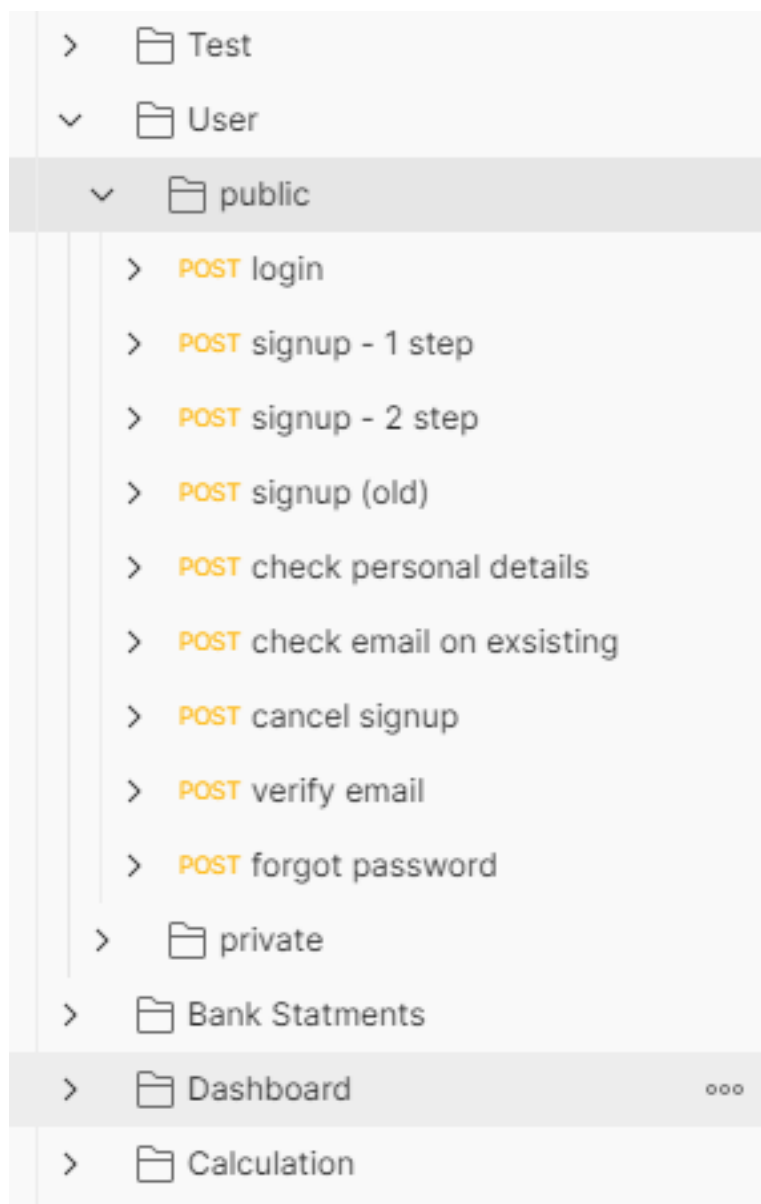


Рисунок 3.3 – Структура розробленого програмного проєкту

Каталог «Test» містить запити, які знаходяться на етапі розробки та випробувань, також можуть стосуватись нового функціоналу, який ще знаходиться на етапі розробки та тестування, також містить чорнові тести та запити.

«User» містить запити які стосуються користувачів їх даних та профілю, це і логін який містить поля логіну та паролю, і верифікація даних введених користувачем при реєстрації, перевірка даних електронної пошти, отримання даних з сервера відповідного юзера для того щоб клієнтська програма могла відобразити ці данні на екрані профіля.

«Bank Statements» включає в себе отримання даних від банку, перевірка коректності введених банківських даних для здійснення покупок.

«Dashboard» запити для отримання даних від сервера для відображення головного екрану застосунку, на головному екрані відображаються різні групи які користувач може купувати.

«Calculation» містить в себе запит розрахункових операцій, вхідні данні відправляються на сервер, і відповідно до введених даними на сервері виконуються розрахунки та надходить відповідь.

3.3 Розробка запитів до серверу

Всі запити до сервера будуються згідно з загальної структури запиту, яка біла зображена на рисунку 1.3. Кожний каталог в проєкті включає в себе запити, які стосуються логічно до цього каталогу. У таблиці 3.1 наведені запити з каталога «User» та їх детальний опис. Тіло запиту наводиться тільки для першого запиту, оскільки тіло в цьому запиті маленьке, для всіх інших запитів з таблиці тіло наведено в додатку А. Таким чином, детальніше запити з цього каталогу та деякі інші запити до сервера, які містять сам запит і тіло запиту, а також відповіді на ці запити наводяться у додатку А.

Таблиця 3.1 – Запити з колекції «User»

Запит	Опис
POST/{{DEV_URL}}/users/login <pre>{ "login": "test@gmail.com", "password": "Secure_123" }</pre>	Запит передає дані, які знаходяться в тілі запиту, на сервер, і сервер відповідно повертає відповідь успішний логін чи ні, та додаткові дані. Змінна DEV_URL містить адресу вебAPI.
POST/{{DEV_URL}}/users	Передає першу частину даних нового юзера до сервера, сервер перевіряє наявність такого юзера в базі, та повертає відповідь. Якщо запит успішний то користувач отримує id для подальшого використання в запитах.
POST/{{DEV_URL}}/users/{{user_id}}/signup	Запит передає додаткові дані за вже існуючим id юзера до сервера.
POST/{{DEV_URL}}/users/check-email	Запит який в тілі надсилає e-mail до сервера, де сервер перевіряє його наявність у базі.
GET/{{DEV_URL}}/users/{{user_id}}/profile	Метод надсилає до серверу запит з id юзера і за цим id отримує від бази даних данні цього юзера, для відображення на екрані профіля.

3.4 Висновки до розділу 3

В цьому розділі було розроблено загальну архітектуру проєкту, та наведено покроковий загальний алгоритм роботи програмного проєкту, який включає в себе два основних компоненти між якими відбувається взаємодія, а саме скрипти запити написані у Postman та вебAPI на стороні серверу.

Також наведено структура розробленого програмного проєкту з описом вмісту всіх розроблених каталогів. Також наводиться приклад запитів одного з каталогів та детальний опис цих запитів.

4 КЕРІВНИЦТВО З ВИКОРИСТАННЯ ПРОГРАМНОГО ПРОДУКТУ

4.1 Призначення програмного продукту

Програмний продукт представляє собою структурований набір запитів написаних на мові JavaScript, який призначений для тестування API на стороні сервера до того як клієнтська частина програми буде розроблена, вона дозволяє отримувати та надсилати дані до сервера використовуючи технологію REST API.

Програмний продукт може використовуватись та бути налаштований на операційних системах таких як Mac та Windows та Linux, на яких можна встановити середовище Postman та з можливістю підключення до мережі Інтернет.

4.2 Вхідні та вихідні дані програмного продукту

Програмний продукт виконує такі функції:

- відправляє дані у форматі HTTP-запитів;
- отримує відповідь від сервера;
- виводить помилки при виникненні збоїв у роботі програми.

Вхідні дані – JSON відповідь, яка має вигляд структурованого тексту, який написано мовою JavaScript, відповідь приходить з різними параметрами залежно від надісланого запиту.

У якості вихідних даних до серверу відправляється запити, тіло запиту також написано мовою JavaScript.

Користувач, за допомогою інтерфейсу програми Postman отримує відповіді від серверу.

4.3 Використання програмного продукту

Середовище Postman використовується як заміну клієнтського застосунку, який би відправляв та отримував дані від сервера.

Перш за все необхідно запуснути застосунок Postman, як тільки він буде встановлено на комп'ютері.

Далі необхідно імпортувати колекцію запитів, яка зберігається у форматі JSON. Для цього в меню необхідно обрати File->Import, та обрати файл «Ugo 1.1.json» (рис. 4.1).

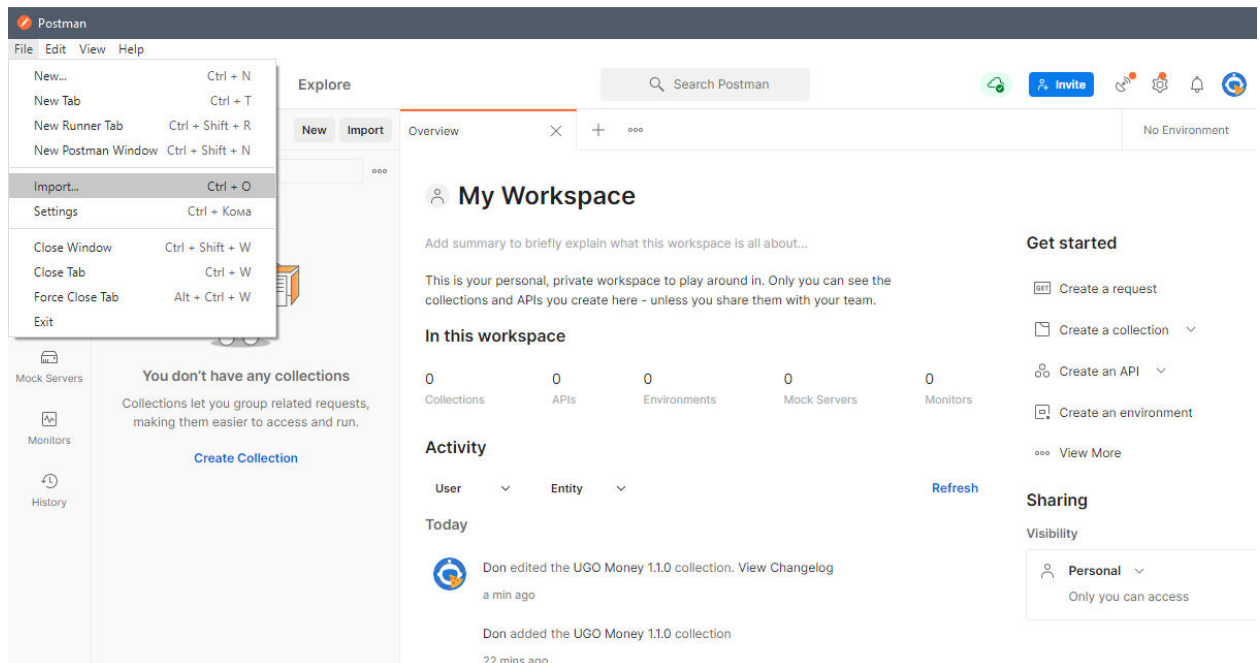


Рисунок 4.1 – Імпорт розробленої колекції в Postman

Після чого в середовищі Postman з'явиться колекція запитів. Далі необхідно відкрити, наприклад, колекцію User. Для прикладу створимо запит login. Для цього необхідно натиснути на колекції User правою кнопкою миші та обрати Add Request. Після чого зліва з'явиться форма запиту, де необхідно написати ім'я запиту «login», з випадаючого списку обрати метод POST, та написати посилання на вебAPI тестового сервера у полі Request URL (рис. 4.2).

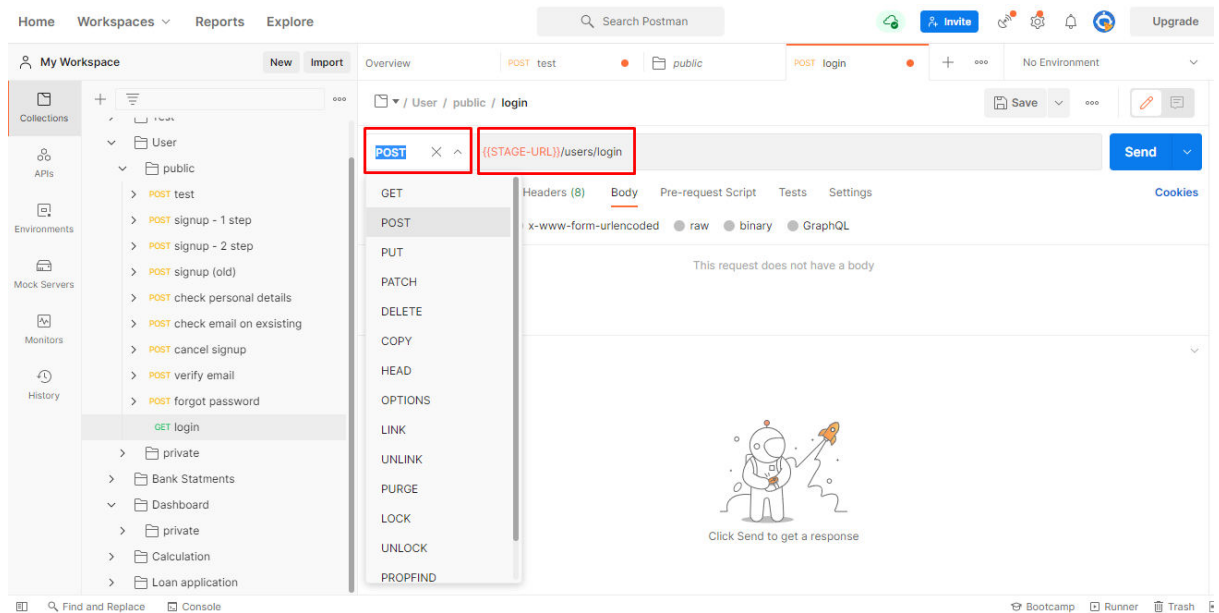


Рисунок 4.2 – Вибір методу запиту та налаштування рядка стану

Після чого необхідно написати тіло запиту, для цього треба перейти на вкладку Body. Встановити потрібний тип вмісту Body та представлення редактора, наприклад, тип вмісту Body - raw, та формат JSON (рис. 4.3).

У текстовому полі, що з'явилося, необхідно написати тіло запиту API у формі пар ключ-значення, в даному випадку це буде логін з визначеною електронною поштою та пароль, наприклад `{"login": "catherine_ferguson@gmail.com", "password": "Secure_123"}`.

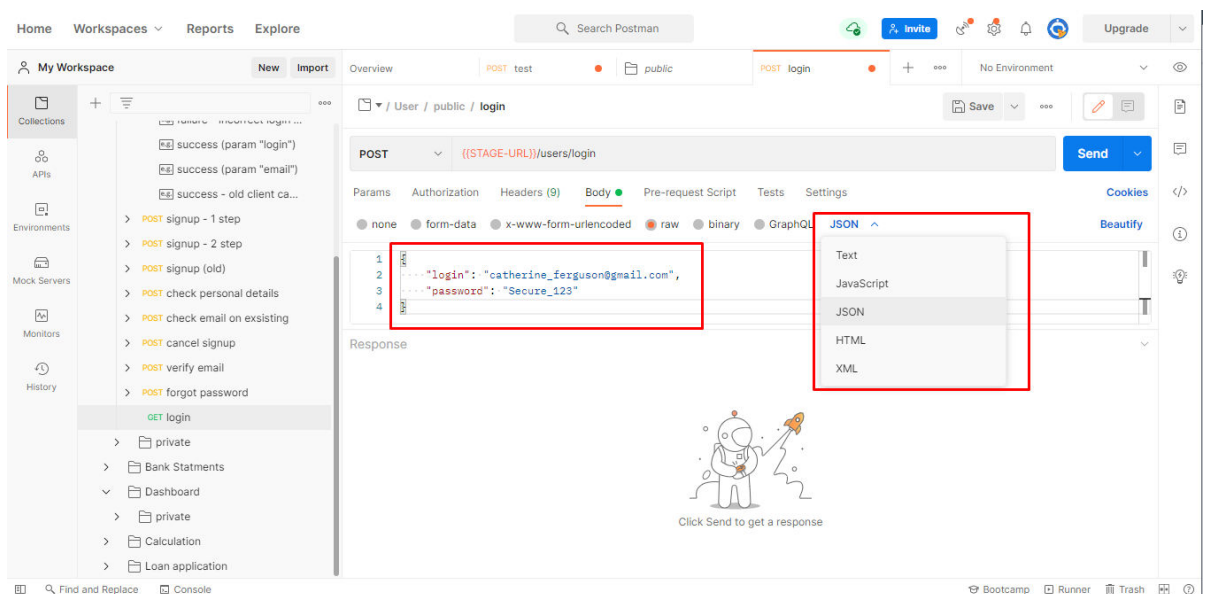


Рисунок 4.3 – Створення тіла запиту

Після чого необхідно зберегти запит у колекції для цього натиснути на кнопку Save (рис. 4.4).

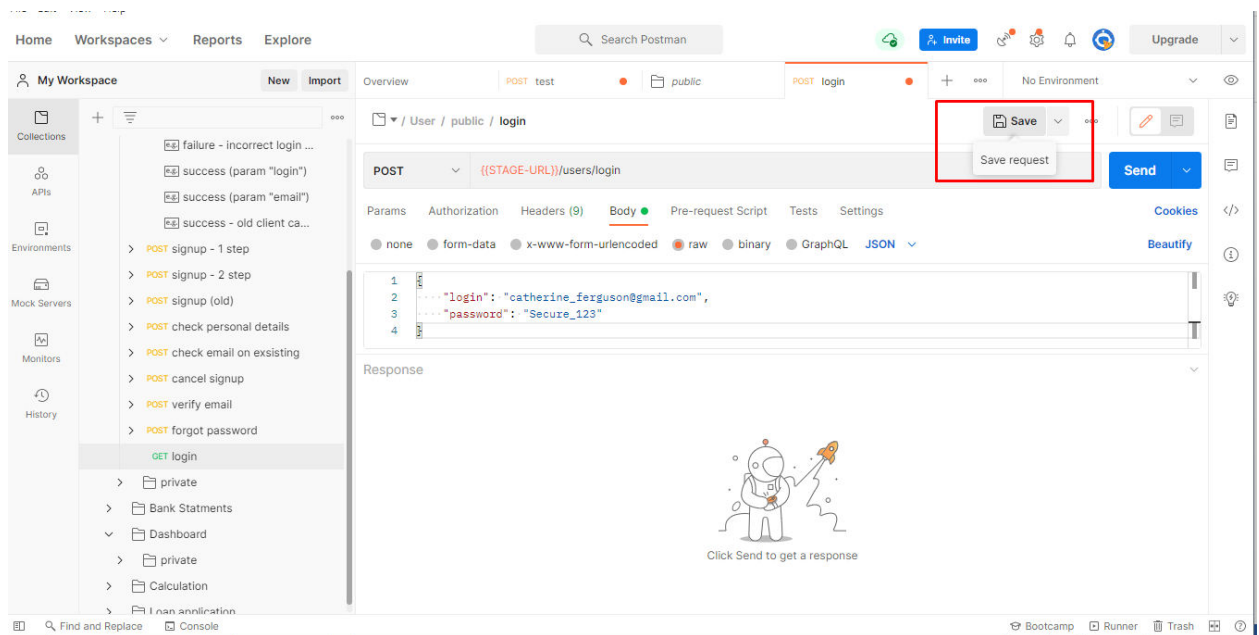


Рисунок 4.4 – Збереження запиту в колекції

Щоб відправити запит до сервера та отримати відповідь від нього необхідно натиснути кнопку Send. Після чого результат запиту з'явиться у нижній частині екрану середовища Postman (рис. 4.5).

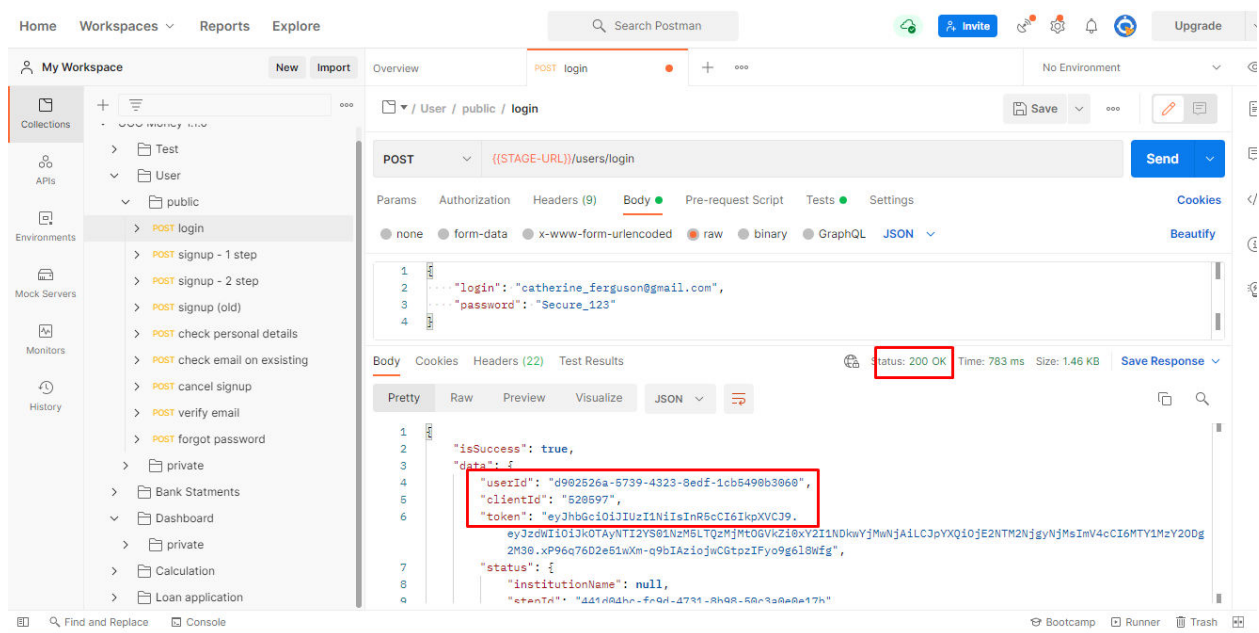


Рисунок 4.5 – Отримана відповідь на запит в Postman

В основному, для тестування вебАРІ нам необхідно перевірити код відповіді, повідомлення у відповідь і тіло відповіді.

Нижче наведено код відповіді, з якою можна зіткнутися під час тестування АРІ.

4.4 Висновки до розділу 4

В даному розділі описано призначення програми, вхідні та вихідні дані. Також наводиться докладна інструкція з використання програмного продукту, детально та покроково описано простий приклад створення запиту, та як відправляти та отримувати результати від сервера.

ВИСНОВКИ

Тестування API являє собою тестування програмного забезпечення, яке зосереджується на тому, щоб визначити, чи відповідають вбудовані API очікуванням щодо функціональності, надійності, продуктивності та безпеки програми.

В ході виконання дипломної кваліфікаційної роботи бакалавра було проведено аналіз та дослідження етапів автоматизації тестування та методів тестування API, та проаналізовано структуру запитів до серверу.

Успішне впровадження практик автоматизації вимагає використання багатьох інструментів для керування конфігурацією, інтеграції, тестування та розгортання. Дуже важливо, щоб інструменти були легко і плавно інтегровані в повний ланцюг інструментів, щоб дозволити розробникам, тестувальникам та іншим зацікавленим сторонам співпрацювати.

Також у цій роботі представлено огляд 3 популярних і перспективних інструментів для автоматизації тестування API шляхом порівняння SoapUI з Postman і Katalon Studio. Кожен засіб має свої плюси і мінуси; але вони широко використовуються розробниками та тестувальниками для тестування API.

Таким чином, проведено аналіз та обґрунтовано вибір середовища автоматизації тестування API. Після проведеного дослідження було обрано середовище Postman, тому що необхідно буде запускати ці тести досить часто і час буде дуже критичним. Цей інструмент є швидким та простим в освоєнні, в основі запитів лежить JSON що також було перевагою, а також безкоштовної версії продукту є досить для роботи. До того ж можливість згенерувати документацію зіграла не останню роль.

Було розроблено загальну архітектуру проєкту, та наведено покроковий загальний алгоритм роботи програмного проєкту, який включає в себе два основних компоненти між якими відбувається взаємодія, а саме скрипти запити написані у Postman та вебAPI на стороні серверу.

Також наведено структура розробленого програмного проєкту з описом вмісту всіх розроблених колекцій. Також наводиться приклад запитів одної з колекції та детальний опис цих запитів.

Таким чином, усі поставлені завдання до дипломної кваліфікаційної роботи бакалавра було виконано.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. What is automation testing? [Electronic source]. – Access mode: <https://www.globalapptesting.com/blog/what-is-automation-testing>.
2. Особенности тестирования Web. Архитектура клиент-сервер - презентация онлайн [Электрон. ресурс]. – Режим доступа: <https://ppt-online.org/490093>.
3. Вебсервер [Электрон. ресурс]. – Режим доступа: <https://uk.wikipedia.org/wiki/Вебсервер>.
4. Уніфікований ідентифікатор ресурсів [Електрон. ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Уніфікований_ідентифікатор_ресурсів.
5. Вебтестирование, особенности, виды, последовательность [Электрон. ресурс]. – Режим доступа: <https://intellect.icu/вебtestirovanie-osobennosti-vidy-posledovatelnost-5191>.
6. HTTP [Электрон. ресурс]. – Режим доступа: <https://uk.wikipedia.org/wiki/HTTP>.
7. Особенности тестирования вебприложений [Электрон. ресурс]. – Режим доступа: <https://quality-lab.ru/blog/key-principles-of-вебtesting/>.
8. REST [Электрон. ресурс]. – Режим доступа: <https://uk.wikipedia.org/wiki/REST>.
9. 15 инструментов для тестирования API [Электрон. ресурс]. – Режим доступа: <https://software-testing.ru/library/testing/functional-testing/2676-open-source-api-testing>.
10. SoapUI, The World's Most Popular API Testing Tool [Electronic source]. – Access mode: <https://www.soapui.org/>.
11. Top 15 API Testing Tools in 2022 (Details & Updates Done!) [Electronic source]. – Access mode: <https://alicealdaine.medium.com/top-10-api-testing-tools-rest-soap-services-5395cb03cfa9>.

12. Katalon, Simplify Web, API, Mobile, Desktop Automated Tests [Electronic source]. – Access mode: <https://katalon.com>.
13. SoapUI vs Postman, Katalon Studio: A Review of Top 3 API Tools | Updated 2022 [Electronic source]. – Access mode: <https://katalon.com/resources-center/blog/soapui-vs-postman-katalon-api-tools>.
14. Postman API Platform, Sign Up for Free [Electronic source]. – Access mode: <https://www.getpostman.com/>.
15. Автоматизация тестирования API, или Почему я решил выбрать Postman [Электрон. ресурс]. – Режим доступа: <https://dou.ua/lenta/articles/choosing-postman/>.

ДОДАТОК А
Текст програми

POST/{{STAGE_URL}}/users/login

```
{
  "login": "catherine_ferguson@gmail.com",
  "password": "Secure_123"
}

{
  "isSuccess": true,
  "data": {
    "userId": "d902526a-5739-4323-8edf-1cb5490b3060",
    "clientId": "520597",
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJkOTAyNTI2YS01NzM5LTQzMjMtOGVhZi0xY2I1NDkwYjMwNjAiLCJpYXQiOiE2NTM2Njk0NTMsImV4cCI6MTY1MzY3MDA1M30u2jZ5mY96D8ismn4R_IHax-Yo4JOdDINsm4EorUai_3E",
    "status": {
      "institutionName": null,
      "stepId": "441d04bc-fc9d-4731-8b98-50c3a0e0e17b",
      "inProgress": false,
      "isCentrelink": true,
      "isWages": true,
      "loanApplication": null
    },
    "isTemp": false,
    "version": "1.2.0"
  }
}
```

```
let res = pm.response.json();
```

```
pm.collectionVariables.set('auth_token', res.data.token);
```

```
pm.collectionVariables.set('user_id', res.data.userId);
```

POST/{{STAGE_URL}}/users

```
{
  "personalDetails": {
    "firstName": "Catherine",
    "lastName": "Ferguson",
    "middleName": "",
    "dateOfBirth": "1990-10-29"
  },
  "contactDetails": {
```

```

        "emailAddress": "catherine_ferguson6@gmail.com",
        "mobileNumber": "0411111111",
        "homeAddress": {
            "streetNumber": 20,
            "streetName": "York",
            "streetType": "street",
            "suburb": "Sydney",
            "state": "NSW",
            "postCode": "2000"
        },
        "stepId": "441D04BC-FC9D-4731-8B98-50C3A0E0E17B"
    }

    {
        "isSuccess": false,
        "error": {
            "code": "",
            "message": "User with email: catherine_ferguson6@gmail.c
om already exists"
        }
    }
}

```

POST/{{STAGE_URL}}/users/check-personal-details

```

{
    "firstName": "Catherine",
    "lastName": "Ferguson",
    "middleName": "",
    "dateOfBirth": "1990-10-29"
}

{
    "isSuccess": true,
    "data": {
        "isRegistered": true,
        "userId": null
    }
}

```

POST/{{DEV_URL}}/users/check-email

```

{
    "email": "catherine_ferguson@gmail.com"
}

```

```

}

{
  "isSuccess": true,
  "data": {
    "isRegistered": true,
    "userId": "d902526a-5739-4323-8edf-1cb5490b3060"
  }
}

```

POST/{{DEV_URL}}/users/verify-email

```

{
  "fullName": "Catherine Ferguson",
  "email": "catherine_ferguson@gmail.com"
}

```

```

{
  "isSuccess": true,
  "data": {
    "code": "643812"
  }
}

```

POST/{{DEV_URL}}/users/forgot-password

```

{
  "login": "jimmybailey@gmail.com"
}

```

```

{
  "isSuccess": false,
  "error": {
    "code": "MULTIPLE_CLIENTS_WITH_THE_SAME_EMAILS",
    "message": "We have identified multiple clients with this email address, please reset password in using your client ID. If you do not have your client ID, please get in touch with Ugo."
  }
}

```

GET/{{DEV_URL}}/users/{{user_id}}/status

```

{
  "isSuccess": true,

```

```

    "data": {
      "institutionName": null,
      "stepId": "441d04bc-fc9d-4731-8b98-50c3a0e0e17b",
      "inProgress": false,
      "isCentrelink": true,
      "isWages": true
    }
  }
}

```

GET/{{DEV_URL}}/users/{{user_id}}/profile

```

{
  "isSuccess": true,
  "data": {
    "fullName": " Catherine Ferguson",
    "email": "catherine_ferguson@gmail.com",
    "phone": "0411111111",
    "address": "20 York Street, Sydney, NSW 2000",
    "created": "2021-12-08T10:01:25Z"
  }
}

```

GET/ {{STAGE_URL}}/users/{{user_id}}/dashboard

```

{
  "isSuccess": true,
  "data": [
    {
      "accountId": "A27444",
      "loanApplicationId": "fdf7952b-1ca5-46c7-8843-f43362be964f",
      "accountStatus": "pre-approved",
      "balance": 0,
      "balanceOpening": 500,
      "closedAmount": 0,
      "dateClosed": null,
      "dateOpened": "2021-10-01T00:00:00Z",
      "nextDue": "2021-10-08T00:00:00Z",
      "particularId": "11",
      "passedTransactions": 0,
      "paymentCycle": "weekly",
      "paymentRegular": 30,
      "totalTransactions": 24,

```

```

        "transactions": null,
        "eSigned": false
    },
    {
        "accountId": "A27481",
        "loanApplicationId": "2ce4cd1f-31fe-4add-ab72-a2dc3d037963",
        "accountStatus": "pre-approved",
        "balance": 0,
        "balanceOpening": 500,
        "closedAmount": 0,
        "dateClosed": null,
        "dateOpened": "2021-11-11T00:00:00Z",
        "nextDue": "2021-11-18T00:00:00Z",
        "particularId": null,
        "passedTransactions": 0,
        "paymentCycle": "weekly",
        "paymentRegular": 30,
        "totalTransactions": 24,
        "transactions": null,
        "eSigned": false
    },
    {
        "accountId": "A27482",
        "loanApplicationId": "aeffb464-d48d-488d-b50f-c6e6f3e0de4f",
        "accountStatus": "pre-approved",
        "balance": 0,
        "balanceOpening": 500,
        "closedAmount": 0,
        "dateClosed": null,
        "dateOpened": "2021-11-11T00:00:00Z",
        "nextDue": "2021-11-18T00:00:00Z",
        "particularId": null,
        "passedTransactions": 0,
        "paymentCycle": "weekly",
        "paymentRegular": 30,
        "totalTransactions": 24,
        "transactions": null,
        "eSigned": false
    },
    {
        "accountId": "A27483",

```

```

        "loanApplicationId": "130cb8aa-7106-48ff-92c8-
4e6064c0b803",
        "accountStatus": "pre-approved",
        "balance": 0,
        "balanceOpening": 500,
        "closedAmount": 0,
        "dateClosed": null,
        "dateOpened": "2021-11-11T00:00:00Z",
        "nextDue": "2021-11-18T00:00:00Z",
        "particularId": null,
        "passedTransactions": 0,
        "paymentCycle": "weekly",
        "paymentRegular": 30,
        "totalTransactions": 24,
        "transactions": null,
        "eSigned": false
    },
    {
        "accountId": "A27484",
        "loanApplicationId": "d413530d-e7aa-4724-9534-
cac388fbelc2",
        "accountStatus": "pre-approved",
        "balance": 0,
        "balanceOpening": 500,
        "closedAmount": 0,
        "dateClosed": null,
        "dateOpened": "2021-11-11T00:00:00Z",
        "nextDue": "2021-11-18T00:00:00Z",
        "particularId": null,
        "passedTransactions": 0,
        "paymentCycle": "weekly",
        "paymentRegular": 30,
        "totalTransactions": 24,
        "transactions": null,
        "eSigned": false
    },
    {
        "accountId": "A27485",
        "loanApplicationId": "7a04919c-3f4d-4cdf-8a5e-
64f212ec4915",
        "accountStatus": "pre-approved",
        "balance": 0,
        "balanceOpening": 500,

```

```

        "closedAmount": 0,
        "dateClosed": null,
        "dateOpened": "2021-11-11T00:00:00Z",
        "nextDue": "2021-11-18T00:00:00Z",
        "particularId": null,
        "passedTransactions": 0,
        "paymentCycle": "weekly",
        "paymentRegular": 30,
        "totalTransactions": 24,
        "transactions": null,
        "eSigned": false
    },
    {
        "accountId": "A27486",
        "loanApplicationId": "5a72df2f-e137-49eb-a4e1-
b9f7b9ebf6dd",
        "accountStatus": "pre-approved",
        "balance": 0,
        "balanceOpening": 500,
        "closedAmount": 0,
        "dateClosed": null,
        "dateOpened": "2021-11-11T00:00:00Z",
        "nextDue": "2021-11-18T00:00:00Z",
        "particularId": null,
        "passedTransactions": 0,
        "paymentCycle": "weekly",
        "paymentRegular": 30,
        "totalTransactions": 24,
        "transactions": null,
        "eSigned": false
    },
    {
        "accountId": "A27487",
        "loanApplicationId": "432c9389-19ef-4872-9444-
27b799919fc6",
        "accountStatus": "pre-approved",
        "balance": 0,
        "balanceOpening": 500,
        "closedAmount": 0,
        "dateClosed": null,
        "dateOpened": "2021-11-11T00:00:00Z",
        "nextDue": "2021-11-18T00:00:00Z",
        "particularId": null,

```

```

    "passedTransactions": 0,
    "paymentCycle": "weekly",
    "paymentRegular": 30,
    "totalTransactions": 24,
    "transactions": null,
    "eSigned": false
  },
  {
    "accountId": "A27488",
    "loanApplicationId": "58665fb5-9db2-4137-99cb-
451aec1c4812",
    "accountStatus": "pre-approved",
    "balance": 0,
    "balanceOpening": 500,
    "closedAmount": 0,
    "dateClosed": null,
    "dateOpened": "2021-11-11T00:00:00Z",
    "nextDue": "2021-11-18T00:00:00Z",
    "particularId": null,
    "passedTransactions": 0,
    "paymentCycle": "weekly",
    "paymentRegular": 30,
    "totalTransactions": 24,
    "transactions": null,
    "eSigned": false
  },
  {
    "accountId": "A27489",
    "loanApplicationId": "2eaafec8-76af-421f-bebb-
49f0ad3d6fd5",
    "accountStatus": "pre-approved",
    "balance": 0,
    "balanceOpening": 500,
    "closedAmount": 0,
    "dateClosed": null,
    "dateOpened": "2021-11-11T00:00:00Z",
    "nextDue": "2021-11-18T00:00:00Z",
    "particularId": null,
    "passedTransactions": 0,
    "paymentCycle": "weekly",
    "paymentRegular": 30,
    "totalTransactions": 24,
    "transactions": null,

```

```

        "eSigned": false
    },
    {
        "accountId": "A27505",
        "loanApplicationId": "1f83a82c-3323-4fdc-9ed0-7d8a50aa87fc",
        "accountStatus": "pre-approved",
        "balance": 0,
        "balanceOpening": 500,
        "closedAmount": 0,
        "dateClosed": null,
        "dateOpened": "2021-11-16T00:00:00Z",
        "nextDue": "2021-11-23T00:00:00Z",
        "particularId": null,
        "passedTransactions": 0,
        "paymentCycle": "weekly",
        "paymentRegular": 30,
        "totalTransactions": 24,
        "transactions": null,
        "eSigned": false
    },
    {
        "accountId": "A27506",
        "loanApplicationId": "58a0fdd2-b1ef-4188-8fda-9880b19719b7",
        "accountStatus": "pre-approved",
        "balance": 0,
        "balanceOpening": 500,
        "closedAmount": 0,
        "dateClosed": null,
        "dateOpened": "2021-11-16T00:00:00Z",
        "nextDue": "2021-11-23T00:00:00Z",
        "particularId": null,
        "passedTransactions": 0,
        "paymentCycle": "weekly",
        "paymentRegular": 30,
        "totalTransactions": 24,
        "transactions": null,
        "eSigned": false
    },
    {
        "accountId": "A27507",

```

```

        "loanApplicationId": "c01f99a5-3807-43d4-b25f-
de9b6b550816",
        "accountStatus": "pre-approved",
        "balance": 0,
        "balanceOpening": 500,
        "closedAmount": 0,
        "dateClosed": null,
        "dateOpened": "2021-11-16T00:00:00Z",
        "nextDue": "2021-11-23T00:00:00Z",
        "particularId": null,
        "passedTransactions": 0,
        "paymentCycle": "weekly",
        "paymentRegular": 30,
        "totalTransactions": 24,
        "transactions": null,
        "eSigned": false
    },
    {
        "accountId": "A27508",
        "loanApplicationId": "bc11a832-71b9-429d-88ba-
616dd16602a9",
        "accountStatus": "pre-approved",
        "balance": 0,
        "balanceOpening": 500,
        "closedAmount": 0,
        "dateClosed": null,
        "dateOpened": "2021-11-16T00:00:00Z",
        "nextDue": "2021-11-23T00:00:00Z",
        "particularId": null,
        "passedTransactions": 0,
        "paymentCycle": "weekly",
        "paymentRegular": 30,
        "totalTransactions": 24,
        "transactions": null,
        "eSigned": false
    },
    {
        "accountId": "A27509",
        "loanApplicationId": "e224e14c-bfea-46c8-b0a0-
88c69cad883d",
        "accountStatus": "pre-approved",
        "balance": 0,
        "balanceOpening": 500,

```

```

        "closedAmount": 0,
        "dateClosed": null,
        "dateOpened": "2021-11-16T00:00:00Z",
        "nextDue": "2021-11-23T00:00:00Z",
        "particularId": null,
        "passedTransactions": 0,
        "paymentCycle": "weekly",
        "paymentRegular": 30,
        "totalTransactions": 24,
        "transactions": null,
        "eSigned": false
    },
    {
        "accountId": "A27511",
        "loanApplicationId": "28e7c783-0a75-480c-8c46-
39b85508a9df",
        "accountStatus": "pre-approved",
        "balance": 0,
        "balanceOpening": 500,
        "closedAmount": 0,
        "dateClosed": null,
        "dateOpened": "2021-11-17T00:00:00Z",
        "nextDue": "2021-11-24T00:00:00Z",
        "particularId": null,
        "passedTransactions": 0,
        "paymentCycle": "weekly",
        "paymentRegular": 155,
        "totalTransactions": 4,
        "transactions": null,
        "eSigned": false
    },
    {
        "accountId": "A27512",
        "loanApplicationId": "af4ffa17-809a-46b9-a0cb-
c60739b91be3",
        "accountStatus": "pre-approved",
        "balance": 0,
        "balanceOpening": 500,
        "closedAmount": 0,
        "dateClosed": null,
        "dateOpened": "2021-11-17T00:00:00Z",
        "nextDue": "2021-11-24T00:00:00Z",
        "particularId": null,

```

```

    "passedTransactions": 0,
    "paymentCycle": "weekly",
    "paymentRegular": 155,
    "totalTransactions": 4,
    "transactions": null,
    "eSigned": false
  },
  {
    "accountId": "A27513",
    "loanApplicationId": "4d510fa6-f8dc-473d-a453-
32999382f923",
    "accountStatus": "pre-approved",
    "balance": 0,
    "balanceOpening": 500,
    "closedAmount": 0,
    "dateClosed": null,
    "dateOpened": "2021-11-17T00:00:00Z",
    "nextDue": "2021-11-24T00:00:00Z",
    "particularId": null,
    "passedTransactions": 0,
    "paymentCycle": "weekly",
    "paymentRegular": 155,
    "totalTransactions": 4,
    "transactions": null,
    "eSigned": false
  },
  {
    "accountId": "A27514",
    "loanApplicationId": "8d916650-68f1-4dbf-a169-
f2c3dbede9f8",
    "accountStatus": "pre-approved",
    "balance": 0,
    "balanceOpening": 500,
    "closedAmount": 0,
    "dateClosed": null,
    "dateOpened": "2021-11-17T00:00:00Z",
    "nextDue": "2021-11-24T00:00:00Z",
    "particularId": null,
    "passedTransactions": 0,
    "paymentCycle": "weekly",
    "paymentRegular": 155,
    "totalTransactions": 4,
    "transactions": null,

```

```

        "eSigned": false
    },
    {
        "accountId": "A27515",
        "loanApplicationId": "2361a9c2-cf49-40f6-aa4e-
5bdaf0d2dfd9",
        "accountStatus": "pre-approved",
        "balance": 0,
        "balanceOpening": 500,
        "closedAmount": 0,
        "dateClosed": null,
        "dateOpened": "2021-11-17T00:00:00Z",
        "nextDue": "2021-11-24T00:00:00Z",
        "particularId": null,
        "passedTransactions": 0,
        "paymentCycle": "weekly",
        "paymentRegular": 155,
        "totalTransactions": 4,
        "transactions": null,
        "eSigned": false
    },
    {
        "accountId": "A27516",
        "loanApplicationId": "ccde80c4-d549-46d6-812a-
5efd084e4f96",
        "accountStatus": "pre-approved",
        "balance": 0,
        "balanceOpening": 500,
        "closedAmount": 0,
        "dateClosed": null,
        "dateOpened": "2021-11-17T00:00:00Z",
        "nextDue": "2021-11-24T00:00:00Z",
        "particularId": null,
        "passedTransactions": 0,
        "paymentCycle": "weekly",
        "paymentRegular": 155,
        "totalTransactions": 4,
        "transactions": null,
        "eSigned": false
    },
    {
        "accountId": "A27517",

```

```
        "loanApplicationId": "c73e3ef5-482b-4b94-82ed-  
f75da3300ef0",  
        "accountStatus": "pre-approved",  
        "balance": 0,  
        "balanceOpening": 500,  
        "closedAmount": 0,  
        "dateClosed": null,  
        "dateOpened": "2021-11-17T00:00:00Z",  
        "nextDue": "2021-11-24T00:00:00Z",  
        "particularId": null,  
        "passedTransactions": 0,  
        "paymentCycle": "weekly",  
        "paymentRegular": 155,  
        "totalTransactions": 4,  
        "transactions": null,  
        "eSigned": false  
    }  
]  
}
```

ДОДАТОК Б
Слайди презентації

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

ДИПЛОМНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА НА ТЕМУ
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТЕСТУВАННЯ API
DEVELOPMENT OF API TESTING SOFTWARE

Виконав:

студент 4 курсу, групи КНТ-218

Шовак К.Г.



Керівник: к.т.н., доц.

Гладкова О.М.

Рисунок Б.1 – Слайд 1

МЕТА ТА ЗАВДАННЯ

Мета роботи – розробити забезпечення для автоматизації тестування серверної частини проєкту, тестування **API**.

Для вирішення поставленої мети необхідно вирішити наступні завдання:

- аналіз можливостей тестування **API**;
- вибір засобів розробки
- розробити структуру проєкту;
- розробити програмне забезпечення.



Рисунок Б.2 – Слайд 2

АЛГОРИТМ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ



3

Рисунок Б.3 – Слайд 3

ЗАГАЛЬНА СТРУКТУРА ЗАПИТУ



4

Рисунок Б.4 – Слайд 4

СТОРОНИ СЕРВЕРУ ТА МЕТОДИ ТЕСТУВАННЯ АРІ



5

Рисунок Б.5 – Слайд 5

АНАЛІЗ ІСНУЮЧИХ HTTP-МЕТОДІВ

Метод	Призначення
Метод GET	Використовується для запиту вмісту на сервері (наприклад, GET /path/resource?param1=value1¶m2=value2 HTTP/1.1).
Метод POST	Даний метод використовується для передачі даних на сервер, проте його основа "ховається" в тіло, що відрізняє його від GET. Під час публікації цієї статті, наприклад, весь текст буде поміщений у тіло POST-запиту; після обробки сервером на сайт буде додано статтю.
Метод HEAD	За своєю суттю не відрізняється від вищезазначеного методу, проте відповідь сервера на такий запит позбавлений «тіла», а практичне застосування орієнтоване на полегшене використання з метою отримання мінімальної інформації про сервер/продукт або його статус.
Метод PUT	Даний метод використовується для заміни одних значень іншими. На відміну від POST він повністю видаляє попереднє значення і встановлює нове, POST в свою чергу дозapisує до вже існуючого значення нове.
Метод DELETE	Видаляє на сервері дані запиту.

6

Рисунок Б.6 – Сайд 6

ОГЛЯД ТИПІВ КОДІВ ВІДПОВІДЕЙ ВІД СЕРВЕРУ

Код	Тип інформації	Призначення
1xx	Інформаційні	Інформування про передачу повідомлення. Самі повідомлення від серверу містять тільки початковий рядок відповіді я, якщо потрібно, декілько специфічних для відповіді полів заголовку.
2xx	Повідомлення про успішну операцію	Інформування про успішну обробку запиту клієнта. Класичним і розповсюдженим є відповідь «200 OK»
3xx	Перенаправляючі	Повідомляють, що для успішного виконання операції необхідно зробити інший запит. Даного типу коди: 301,302, 303, 305, 307. Адреса, за якою клієнту необхідно провести запит, сервер вказує у заголовку Location .
4xx	Помилка на стороні клієнту	Вказує, що помилка на стороні клієнта, тобто в тілі запиту.
5xx	Помилка на стороні серверу	Вказує на помилку виконання операції за вини веб-сервера.

7

Рисунок Б.7 – Слайд 7

ПОРІВНЯННЯ ІНСТРУМЕНТІВ ДЛЯ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ API

Характеристика	SoapUI	Postman	Katalon
Тестування REST API	+	+	+
Автоматична генерація інструкцій	+	+	+
Керування даними	JSON, CVS	JSON, CVS	Excel, JSON, CVS
Тестові звіти	HTML	JSON, HTML	HTML
Мова написання скриптів	Groovy	JS	Groovy, Java
Ціна	30 днів безкоштовно - 755\$ на рік	безкоштовно - 12 \$ /місяць	безкоштовно — 1899 \$ /рік

8

Рисунок Б.8 – Слайд 8

РОЗРОБЛЕНА ЗАГАЛЬНА АРХІТЕКТУРА ТА АЛГОРИТМ РОБОТИ ПРОЄКТУ

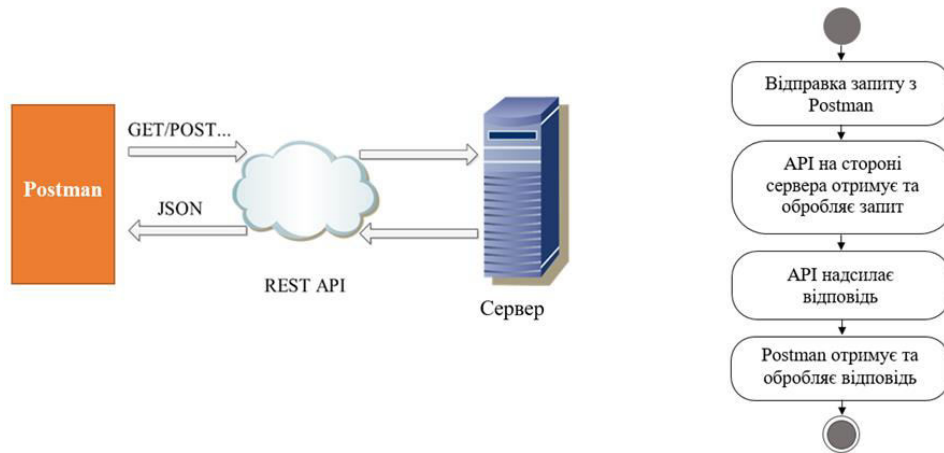


Рисунок Б.9 – Слайд 9

СТРУКТУРА РОЗРОБЛЕНОГО ПРОГРАМНОГО ПРОЄКТУ

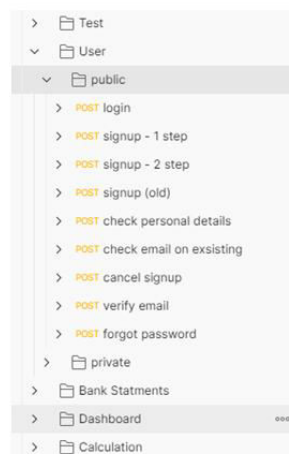


Рисунок Б.10 – Слайд 10

РОЗРОБЛЕНІ ЗАПИТИ З КОЛЕКЦІЇ «USER»

Запит	Опис
POST /{(DEV_URL)}/users/login { "login":"test@gmail.com", "password":"Secure_123"}	Запит передає дані, які знаходяться в тілі запити, на сервер, і сервер відповідно поверає відповідь успішній логін чи ні, та додаткові дані. Змінна DEV_URL містить адресу веб-API.
POST /{(DEV_URL)}/users	Передає першу частину дани нового юзера до сервера, сервер перевіряє наявність такого юзера в базі, та поверає відповідь. Якщо запит успішний то юзер отримує id для подальшого використання в запитах.
POST /{(DEV_URL)}/users/{(user_id)}/signup	Запит передає додаткові дані за вже існуючим id юзера до сервера.
POST /{(DEV_URL)}/users/check-email	Запит який в тілі надсилає e-mail до сервера, де сервер перевіряє його наявність у базі.
GET /{(DEV_URL)}/users/{(user_id)}/profile	Метод надсилає до серверу запит з id юзера і за цим id отримує від бази даних данні цього юзера, для відображення на екрані профіля.

Рисунок Б.11 – Слайд 11

ПРИКЛАД РОБОТИ РОЗРОБЛЕНОГО ПРОГРАМНОГО ПРОДУКТУ

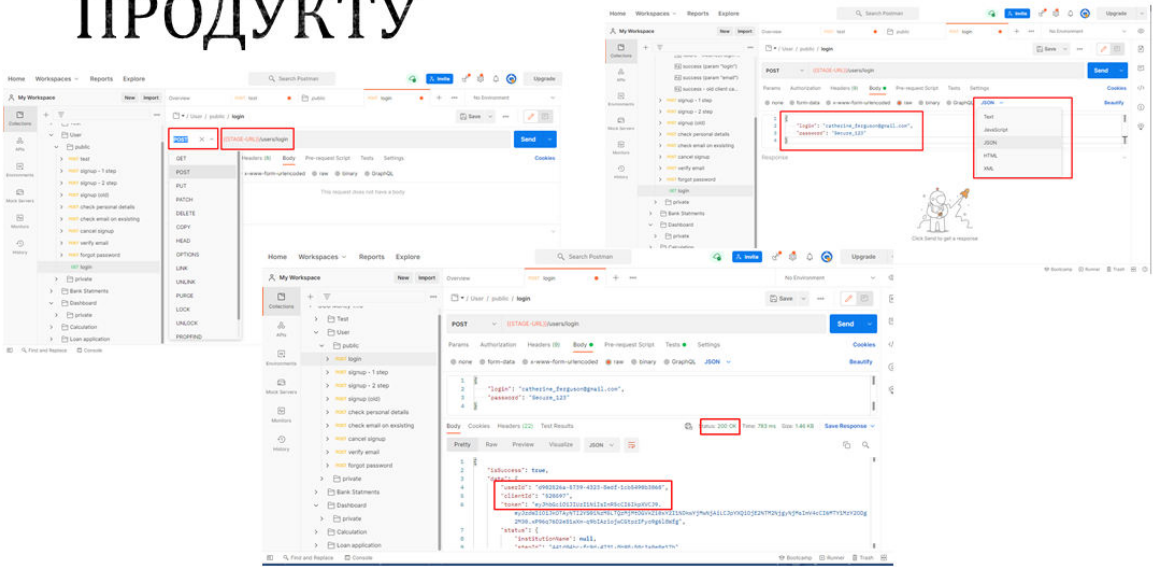


Рисунок Б.12 – Слайд 12

ВИСНОВКИ

В ході виконання дипломної кваліфікаційної роботи бакалавра було:

- Проведено аналіз та дослідження етапів автоматизації тестування та методів тестування API, та проаналізовано структуру запитів до сервера.
- Проведено аналіз та обґрунтовано вибір середовища автоматизації тестування API.
- Було розроблено загальну архітектуру проєкту, та наведено покроковий загальний алгоритм роботи програмного проєкту, який включає в себе два основних компоненти між якими відбувається взаємодія, а саме скрипти запити написані у Postman та веб-API на стороні серверу.
- Також наведено структура розробленого програмного проєкту з описом вмісту всіх розроблених каталогів. Також наводиться приклад запитів одного з каталогів та детальний опис цих запитів.



Рисунок Б.13 – Слайд 13