Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інтелектуальних інформаційних технологій та автоматизації
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Інформаційна технологія автоматизованого тестування WEB-додатків. Серверна частина»

Виконав: студент 2-го курсу, групи 2КН-21м
спеціальності 122 – Комп'ютерні науки
_____ Галка О.К.
(прізвище та ініціали)

Керівник: д-р техн. наук, доцент, проф.,
_____ Іванчук Я.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
«15» 12 2022 р.

Опонент: к.т.н., доц. каф. АІІТ:
_____ Коцюбинський В. Ю.
(прізвище та ініціали)
«17» грудня 2022 р.

Допущено до захисту
Завідувач кафедри КН
д.т.н., проф. Яровий А.А.
(прізвище та ініціали)
«16» 12. 2022 р.

Вінниця ВНТУ – 2022 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 122 «Комп'ютерні науки»
Освітньо-професійна програма – Системи штучного інтелекту

ЗАТВЕРДЖУЮ
Завідувач кафедри _____
д.т.н., проф.. Яровий А.А.
"15" 09 2022 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Галці Олександру Костянтиновичу

1. Тема роботи: «Інформаційна технологія автоматизованого тестування WEB-додатків. Серверна частина»
керівник роботи д-р техн. наук., проф., Іванчук Я. В.
затверджені наказом вищого навчального закладу від «14» вересня 2022 року №203.
2. Строк подання студентом роботи 18 листопада 2022 року.
3. Вихідні дані: максимально допустима швидкість транзакцій – 10 од./с; багатозадачність при тестуванні; максимально допустима кількість запитів від користувача – 5 од.; максимально допустима швидкість обробки даних – 50 од./с.; максимально допустима швидкість обміну даних – 125 од./с.; ймовірність виникнення збою роботи програмного модуля – 45%; допустимий час отримання відповіді від сервера – 1 с; степінь хешування персональних даних користувачів – 32 ум. од.
4. Зміст текстової частини: вступ, аналіз сучасного стану розвитку систем автоматизованого тестування WEB-додатків, розробка інформаційної технології автоматизованого тестування WEB-додатків програмна реалізація системи автоматичного тестування WEB-додатків, економічна частина, висновки, список використаних джерел.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень): схема класифікації тестування за принципом роботи, алгоритм автоматизованої побудови тестових сценаріїв, схема алгоритму реєстрації та входу користувача в систему тестувань, діаграма класу «Користувача», діаграма класів Schedules, Users, Reports, показники надійності WEB-додатка, головне меню інтерфейсу сервера, приклад роботи реєстрації та входу, діаграма ймовірності виникнення збою роботи програмного модуля серверної частини від вмісту хешованих даних на сервері, діаграма залежності рівня безпеки персональних даних в степеня хешування.

5. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	виконання прийняв
Спеціальна частина. Розділ 1-3.	[illegible]	14.09.2022р	14.09.2022р
Економічна частина. Розділ 4	Бронніков А.В. д.е.н. професор кафедри ЕПВ	14.09.2022р	14.09.2022р

Дата видачі завдання 14.09 2022 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз сучасного стану розвитку систем автоматизованого тестування WEB-додатків	14.09.2022р – 01.10.2022р
2	Розробка інформаційної технології автоматизованого тестування WEB-додатків	02.10.2022р – 16.10.2022р
3	Розробка серверної системи автоматичного тестування WEB-додатків	17.10.2022р – 07.11.2022р
4	Програмна реалізація системи автоматичного тестування WEB-додатків	08.11.2022р – 21.11.2022р
5	Апробація та/або впровадження результатів дослідження	23.11.2022р – 01.12.2022р
6	Оформлення пояснювальної записки, графічного матеріалу та презентації	02.11.2022р – 14.12.2022р

Студент (підпис) Галка О. К.

Керівник роботи (підпис) Іванчук Я. В.

АНОТАЦІЯ

УДК 004.054

Галка О.К. Інформаційна технологія автоматизованого тестування WEB-додатків. Серверна частина. Магістерська кваліфікаційна робота зі спеціальності 122 – комп'ютерні науки, освітня програма - системи штучного інтелекту. Вінниця: ВНТУ, 2022. 102 с.

На укр. мові. Бібліогр.: 21 назв; рис.: 22; табл. 9.

Дана магістерська кваліфікаційна робота присвячена розробці інформаційної технології та програмного забезпечення для автоматизованого тестування WEB-додатків. Розроблені математична модель «чорної скриньки» для автоматизованого тестування. Також розроблено метод «чорної скриньки» динамічного тестування. Розроблена серверна частина, призначена для автоматизованого тестування. Для розробки проекту використано програмне середовище PyCharm, мова програмування Python. Для розробки прикладного програмного інтерфейсу сервера було використано бібліотеку Flask, для збереження даних використовувалась об'єктно-орієнтована база даних Mongo. Визначено, що використання результатів діагностики дозволяє загалом підвищити надійність функціонування веб-додатків на серверному рівні. Графічна частина складається з 4 плакатів із результатами проектування та реалізації.

Ключові слова: продуктивність; автоматизація; тестування; веб-додаток.

ABSTRACT

Halka O.K. Information technology for automated testing of WEB applications. Server part. Master's thesis on specialty 122 - computer science, educational program - artificial intelligence systems. Vinnytsia: VNTU, 2022. 102 p.
In Ukrainian speech Bibliography: 21 titles; Fig.: 22; table 9.

This Master's thesis is devoted to the development of information technology and software for automated testing of WEB applications. A mathematical model of the "black box" for automated testing was developed. The "black box" method of dynamic testing has also been developed. The server part designed for automated testing has been developed. The PyCharm software environment, the Python programming language, was used to develop the project. The Flask library was used to develop the application software interface of the server, and the Mongo object-oriented database was used to store data. It was determined that the use of diagnostic results allows to generally increase the reliability of the functioning of web applications at the server level. The graphic part consists of 4 posters with the results of design and implementation.

Keywords: reliability; automation; testing; web application.

ЗМІСТ

ВСТУП

Нові методології програмування, такі як прискорена розробка додатків (RAD) і екстремальне програмування призвели до інтенсивного розвитку засобів автоматизованого тестування. Головною особливістю цих методів є можливість отримувати різні версії програмних продуктів, кількість яких інтенсивно збільшується [1]. Як наслідок, сучасне тестування є ітеративним процесом – кожен новий випуск супроводжується наявністю нових ефективних тестів, а також оновленням існуючих автоматизованих інструментів тестування. Унікальністю конструкції автоматизованих засобів тестування є специфіка завдань, які перед ними стоять: об’єктивна складність тестування - це деструктивний, тобто зворотний творчий процес (проектовані засоби повинні замість збору та обробки інформації виконувати її розбиття на частини і проводити аналіз цих частин) [2]; кількість варіантів тестування абсолютно всіх можливих ситуацій дуже велика, тому проектовані засоби тестування повинні враховувати неможливість перебору усіх можливих тестів [3].

Тестування дуже важливий процес із-за непередбачуваної наявності помилок у процесі проєктування і розробки самого програмного забезпечення (ПЗ). Деякі з цих помилок не є критичними, але є такі, що значно дороговартісні і небезпечні для життєдіяльності самої людини. Тому, з метою усунення помилок, необхідно виконувати перевірку абсолютно усіх програмних модулів, що знаходяться в процесі розробки [4].

Тестування ПЗ має багато переваг, однією з яких є його економічна ефективність. Тестування заощаджує значні кошти в довгостроковій перспективі. Розробка ПЗ, а саме WEB-додатків, складається із багатьох етапів, і якщо помилки виявлені на ранніх стадіях то виправити їх у кінці-кінців набагато дешевше. Тому питання, які пов’язані із тестуванням WEB-додатків є актуальним [4].

Це пов’язано з тим, що сучасні WEB-додатки використовуються в усіх сферах життя. Тести продуктивності не обов’язково відображають помилки самої програми [5]. Він має підтримувати WEB-програму в робочому стані незалежно від коливань мережі, доступності пропускної здатності чи навантаження на трафік [6].

Тому розробка та виконання цих тестів має вирішальне значення для забезпечення стабільності роботи WEB-додатку. Наприклад, при відкритті WEB-програми онлайн-банкінгу (сервіс із найнижчою комісією на ринку для швидких грошових переказів) під час «реєстрації» та отримання повідомлення виникає «помилка», тобто відбувається використання WEB-програми, до якої одночасно можуть отримати доступ багато людей, хоча самий доступ до сайту є обмеженим. Відповідно із-за даної проблеми багато користувачів не можете «зареєструватися» і використовувати цей продукт [7], наслідком чого є зрив комерційної угоди і відповідно втрати власних коштів. Користувачеві нічого не залишається, як шукати іншу подібну WEB-програму, яка працює стабільно і надійно. Тестування допомагає заощаджувати час і гроші в довгостроковій перспективі, оскільки проблеми із надійністю і стабільністю вирішуються до того, як виникнуть серйозніші проблеми [8]. Тому актуальним є розробка інформаційної технології автоматизованого тестування WEB-додатків, яка дозволить збільшити ефективність генерації різних видів тестів для підвищення рівня працездатності WEB-додатків.

Зв’язок роботи з науковими програмами, планами, темами. Магістерська робота виконана відповідно до напрямку наукових досліджень кафедри комп’ютерних наук Вінницького національного технічного університету 22 К1 «Моделі, методи, технології та пристрої інтелектуальних інформаційних систем управління, економіки, навчання та комунікацій» та плану наукової та навчально-методичної роботи кафедри.

Мета та завдання дослідження. Метою магістерської кваліфікаційної роботи є підвищення рівня надійності функціонування програмних WEB-

додатків за допомогою розробки інформаційної технології автоматизованого тестування на рівні серверного модуля програмної системи.

Для досягнення поставленої мети необхідно розв'язати такі наступні **задачі**:

- провести аналіз предметної області автоматизованого тестування WEB-додатків;
- провести аналіз відомих технічних рішень серверних модулів програмних систем автоматизованого тестування WEB-додатків;
- провести порівняльний аналіз характеристики клієнтських модулів програмних систем автоматизованого тестування WEB-додатків;
- розробити математичну модель динамічного автоматизованого тестування WEB-додатків;
- розробити метод автоматизованого динамічного тестування WEB-додатків;
- провести проектування інтелектуальної моделі системи автоматизованого тестування WEB-додатків;
- програмно реалізувати систему автоматичного тестування WEB-додатків на клієнтському рівні;
- провести тестування клієнтської частини системи автоматизованого тестування WEB-додатків;
- провести розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним.

Об'єкт дослідження – процес автоматизації тестування WEB-додатків.

Предмет дослідження – програмні засоби автоматизації тестування WEB-додатків.

Методи дослідження - методи та підходи до розробки автоматизованих систем, методи та підходи до розробки WEB-орієнтованих програмних додатків, методи об'єктно-орієнтованого програмування.

Наукова новизна одержаних результатів полягає в наступному: розроблено інформаційну технологію автоматичного тестування WEB-додатків, що відрізняється від існуючих використанням математичної моделі динамічного тестування надійності програмного забезпечення методом «чорної скриньки», що визначає інтенсивність відмов кожної компоненти програмного коду.

Практичне значення одержаних результатів полягає у наступному:

1. Розроблено алгоритм роботи програмного забезпечення серверної частини автоматизованого тестування WEB-додатків.
2. Здійснено програмну реалізацію серверної частини системи автоматизованого тестування WEB-додатків.

Достовірність теоретичних положень магістерської кваліфікаційної роботи підтверджується строгістю постановки задач, коректним застосуванням математичних методів під час доведення наукових положень, строгим виведенням аналітичних співвідношень, порівнянням результатів з відомими, та збіжністю результатів математичного моделювання з результатами, що отримані під час впровадження розроблених програмних засобів.

Особистий внесок магістранта. Усі результати, наведені у магістерській кваліфікаційній роботі, отримані самостійно.

Апробація результатів роботи. Результаті дослідження апробовані на LI Науково-технічній конференції факультету інтелектуальних інформаційних технологій та автоматизації (2022) [1].

Публікації. За результатами дослідження опубліковано одні тези доповідей [1].

1 АНАЛІЗ СУЧАСНОГО СТАНУ РОЗВИТКУ СИСТЕМ АВТОМАТИЗОВАНОГО ТЕТСТУВАННЯ WEB-ДОДАТКІВ

1.1 Аналіз предметної області автоматизованого тестування програмних засобів

Тестування – це одна з технік контролю якості, що включає в себе планування, проектування, виконання тестування та аналіз відповідних результатів.

Тестування програмного забезпечення – це процес аналізу програмного засобу і відповідної документації, з метою виявлення дефектів і підвищення якості продукту [1].

Тестування програмного забезпечення охоплює не тільки проведення тестів, але й інші елементи процесу забезпечення якості, такі як:
планування та аналіз вимог; критерії початку тестування; стратегія тестування; проектування тестових сценаріїв (тест-планів, тест-кейсів, користувацьких вимог); виконання тест-кейсів, вимог; фіксація дефектів; аналіз результатів; написання звітів.

Стосовно цілі тестування, можна виділити такі основні аспекти:

- надання інформації про якість програмного забезпечення кінцевому замовнику;
- підвищення якості тестового ПЗ;
- запобігання виявленню нових дефектів.

До основних типів тестування відноситься:

1) Ручне тестування (Manual Testing) – це вид тестування, в якому тестують ПО вручну, тобто не використовують ніяких середовищ автоматизації.

2) Тестувальник має роль звичайного користувача програми та перевіряє продукт, щоб виявити баги у програмі. Ручне тестування – найбільш низькорівневий і простий тип тестування, який може займати багато часу, але якщо глянути на довгострокову перспективу, то він економічно вигідніший [1].

Ручне тестування поділяється на декілька видів: модульне тестування (Unit Testing); інтеграційне тестування (Integration Testing); системне тестування (System Testing); операційне тестування (Release Testing); приймальне тестування (Acceptance Testing).

Для професійного тестування використовується тест-плани з варіантами тестування. Тест-план – це документ, який описує весь розмір роботи, яку повинен виконати тестувальник, починаючи з опису об’єкту, цілі, ресурси і графіки запланованих тестових активностей, стратегії тестування, розкладу, критерії початку і кінця тестування, до необхідного в процесі тестування обладнання, методи проектування тестів. Оцінка всіх можливих ризиків тестування з варіантами їх можливого вирішення [2].

Переваги такого виду тестування полягає в таких аспектах:

- користувацький зворотній зв’язок, де самий звіт тестувальника може бути розглянутим як відгук від повноцінного користувача;
- Інтерфейсний зворотній зв’язок. Користувацький інтерфейс можна протестувати повністю тільки в ручну;
- економічна вигода;
- Тестування в реальному часі. Можливість розпочати тестування на перших етапах впровадження без кінцевого продукту;
- можливість дослідницького тестування.

Недоліки такого тестування:

- Людський фактор. Це найбільш впливовий фактор, тому що усі люди можуть допустити помилки;
- неможливість перевірки навантажувального тестування;
- важкість повторної перевірки після внесення змін.

Автоматизоване тестування (Automated Testing) – це набір технік, підходів і інструментальних елементів, які дозволяють вилучити тестувальника з виконання деяких завдань в процесі тестування [3].

У автоматизованому тестуванні використовується три рівні:

1. Тестування на рівні коду (модульне тестування);

2. Функціональне тестування;

3. GUI – тестування (Graphical User Interface – Графічний інтерфейс користувача) [4].

Для професійного тестування використовуються тест-кейси, які частково або повністю покривають інструментальне середовище, однак розробка тест-кейсів, запуск, оцінка виконання тестів та опис дефектів в баг-трекінгових системах не обходиться без втручання тестувальника.

Тест-кейс – набір вхідних даних, умови виконання, очікуваний і фактичний результат, розроблений з цілю перевірки властивостей і поведінку програмного середовища. Переваги такого виду тестування: тестування навантаженості системи; час виконання автотестів; легкість повторної перевірки після внесення змін; повторюваність. Недоліки такого тестування: великі витрати; вартість інструменту автоматизації; пропуск дрібних помилок.

Напівавтоматизоване тестування (Semiautomated Testing) – використовуючи цей вид тестування, частину проводимо в ручному тестуванні, а іншу половину ми автоматизуємо за допомогою тестів. Тестування можна класифікувати за дуже великою кількістю ознак.

Розглянемо список спрощеної класифікації тестування (рис. 1.1) [5].

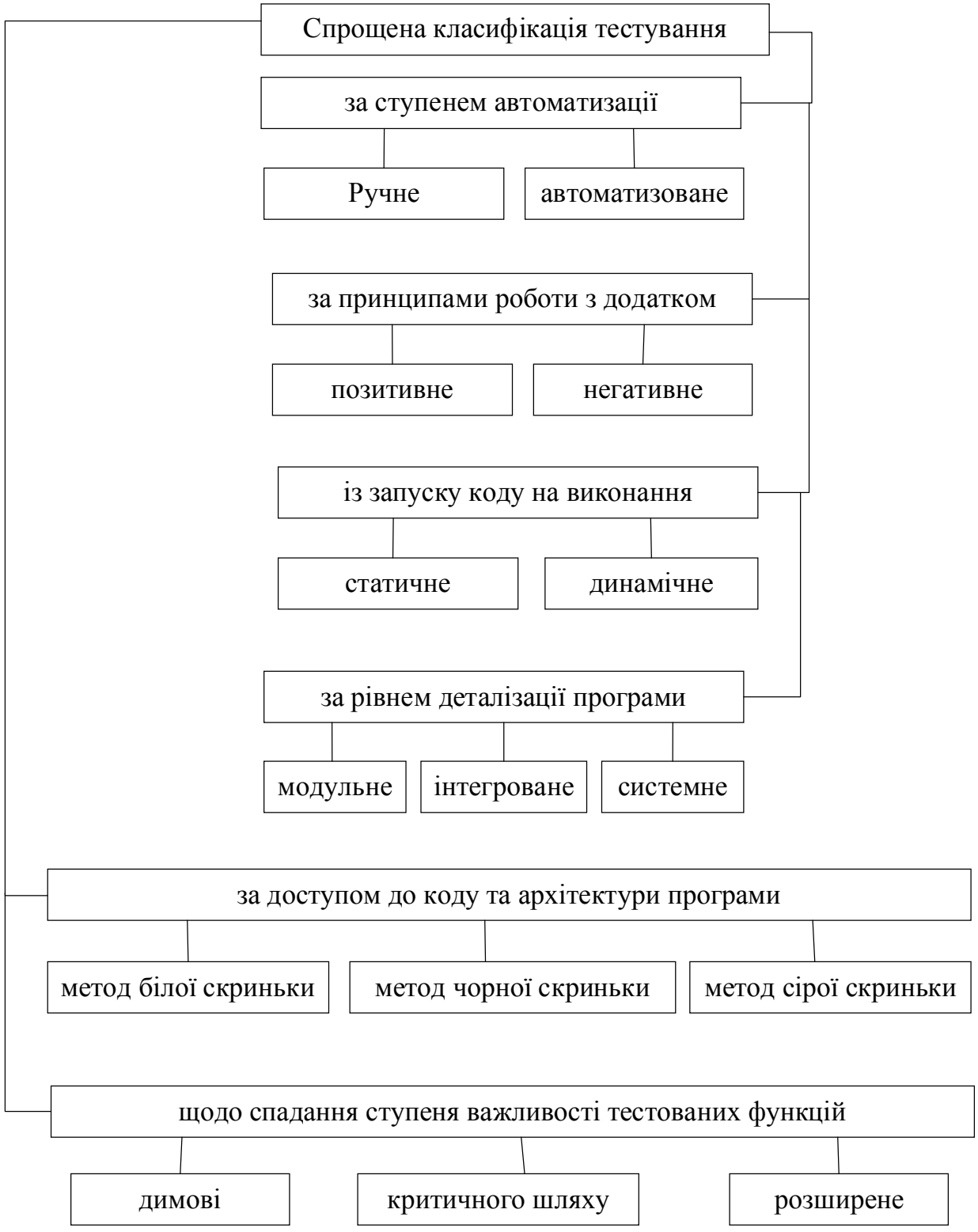


Рисунок 1.1 – Схема класифікації тестування за принципом роботи

Перший вид – це тестування по запуску коду на виконання їх є два це:

- статичне тестування – без запуску коду;

– динамічне тестування – з запуском коду.

До статичного тестування відносяться: документи (вимоги користувацькі, бізнес вимоги, тест-кейси, користувацькі історії, база даних); графічні прототипи; код програми; параметри програми; підготовка тестових даних.

Динамічне тестування передбачає роботу з кодом, подільним кодом, його частинами, перевірку поведінки програми.

Методи доступу до програмного коду й самої програми: метод білого ящика (наявність доступ до програмного коду); метод сірого ящика – тільки до частини коду є доступ; метод чорного ящика – без доступу до коду.

За ступенем автоматизації: ручне тестування; автоматизоване тестування.

За рівнем деталізації додатку:

– Модульне тестування. Перевіряє функціональність і шукає дефекти в частинах додатка, які доступні і можуть бути протестовані окремо;

– Інтеграційне тестування. Перевіряється взаємодія між компонентами системи після проведення компонентного тестування;

– Системне тестування. Перевірка як функціональних, так і нефункціональних вимог в системі вцілому.

За ступенем важливості тестових функцій:

– Димове тестування. Перевіряються ключові функціональності, без яких сам додаток не маєважливості;

– Тестування критичного шляху. Перевіряються найбільш важливі елементи і функції програми, правильністьроботи та їх використання;

– Розширене тестування. Перевірка усієї іншої функціональності.

За принципом роботи з додатком:

– Позитивне тестування. Це тестування, яке перевіряє поведінку нашої програми на звичайне користування. Введення даних, які відповідають дійсності;

– Негативне тестування. Це тестування, яке перевіряє поведінку програми на не звичні для нього команди. Введення даних, які не відповідають

дійсності. Перевірка виводу помилок, і перевірка того, як поведе себе даний додаток.

Класифікація за природою додатка:

– Тестування WEB-додатка. Тестування пов’язане з інтенсивною діяльністю в області тестування сумісності, тестування продуктивності.

– Тестування мобільних додатків. Також вимагає підвищеної уваги до тестування сумісності, оптимізації продуктивності, автоматизації тестування із застосуванням емуляторів мобільних пристроїв;

– Тестування додатків [6]. Є найбільш класичним з цих 3 класифікацій, і його особливості залежать від предметної області додатків, нюансів архітектури, ключових показників якості.

Автоматизація тестів – найкращий спосіб підвищити ефективність, охоплення тестом і швидкість виконання при тестуванні програмного забезпечення [7].

Автоматизоване тестування програмного забезпечення важливо з таких причин: ручне тестування всіх робочих процесів, усіх полів, усіх негативних сценаріїв вимагає часу та грошей; складно перевірити багатомовні сайти вручну; автоматизація тестів при тестуванні програмного забезпечення не вимагає втручання людини. Ви можете запустити тест і залишити його без нагляду; автоматизація тестів збільшує швидкість виконання тесту; автоматизація сприяє збільшенню охоплення тестом; тестування вручну може стати нудним, схильним до помилок [8]. Автоматизований процес тестування:

У процесі автоматизації виконуються наступні кроки зображено на рисунку 1.2: вибір інструменту тестування, визначення сфери автоматизації, планування, проєктування та розробка, виконання тесту, технічне обслуговування.

Рисунок 1.2 – Процеси автоматизованого тестування

Види автоматизованого тестування [9]:

– Димове тестування. Тестування, що проводиться на початковому етапі і в першу чергу спрямоване на перевірку готовності розробленого продукту до проведення більш розширеного тестування, визначення загального стану якості продукту;

– Модульне тестування. Це вид тестування програмного забезпечення, де тестуються окремі блоки або компоненти програмного забезпечення. Метою є перевірити, що кожна одиниця програмного коду працює належним чином. Модульне тестування проводиться під час розробки програми розробниками;

– Інтеграційне тестування. Визначається як тип тестування, де програмні модулі інтегровані логічно та перевіряються як група [5]. Типовий програмний проєкт складається з безлічі програмних модулів, кодованих різними програмістами. Метою цього рівня тестування є виявлення дефектів взаємодії між цими програмними модулями при їх інтеграції;

– Функціональне тестування. Вид тестування, спрямований на перевірку коректності роботи функціональності додатку (коректність реалізації функціональних вимог). Часто функціональне тестування асоціюють з тестуванням методом чорного ящика, однак і методом білого ящика цілком можна перевіряти коректність реалізації функціональності;

– Тестування на основі ключових слів. Спеціалізований підхід, згідно з яким використовуються деякі ключові слова, детально описують набір

виконуваних дій, які, так чи інакше, потрібні для проходження певного етапу тестового сценарію [10]; регресійне тестування визначається як тип тестування програмного забезпечення для підтвердження того, що нещодавня зміна програми або коду не вплинула негативно на існуючі функції [11]; перевірка рівня даних сконцентрована на тій частині програми, яка відповідає за зберігання і деяку обробку даних (найчастіше – в базі даних чи іншому сховищі). Тут особливий інтерес представляє тестування даних, перевірка дотримання бізнес-правил, тестування продуктивності тестування чорної скриньки [12] – це техніка, що використовується для перевірки функціональності програмного забезпечення, що працює виключно із зовнішнім інтерфейсом. Цей метод тестування також називають поведінковим тестуванням та функціональним тестуванням.

WEB-додаток – клієнт-серверний додаток, в якому клієнт взаємодіє з WEB- сервером за допомогою браузера [9].

Програмне забезпечення WEB-додатків складається з програм та даних, призначених для роботи в усьому світі, поєднання мережевого та клієнт-серверного програмного та апаратного забезпечення, виконуючи операції між локальними та віддаленими користувачами комп’ютера.

Для того, щоб найкраще визначити, як тестувати WEB-програмне забезпечення, як і будь-яке інше програмне забезпечення, тестери повинні розуміти характеристики WEB-програмного забезпечення, поведінку загалом та її взаємодію зіншим програмним та апаратним забезпеченням.

Web-програмне забезпечення, як і будь-яке інше програмне забезпечення, має особливий потенціал, проблемні місця, що призведе до збою програми або системи, якщо програма вийде з ладу, нам необхідно передбачити такі місця та впоратись із проблемами.

1.2 Аналіз відомих технічних рішень серверних систем програмних засобів автоматизованого тестування WEB-додатків

Засоби автоматизації тестування – це програмні інструменти, які допомагають користувачам тестувати різні настільні, WEB- та мобільні програми. Ці інструменти надають рішення для автоматизації з метою автоматизації процесу тестування. Автоматизовані засоби тестування також пропонують безліч функцій для тестування графічного інтерфейсу, тестування продуктивності, тестування навантаження та тестування API [10].

Нижче наведено кілька найкращих інструментів автоматизації випробувань: Selenium, Katalon Studio, UFT One.

Ось найкращі засоби автоматизації тестів, які, як вважають, найкраще вирішують проблеми автоматизації протягом кількох років. Інструменти, включені до цього списку, обираються за цими критеріями:

- підтримка тестування API та сервісів;
- пропонуючи деякі можливості AI/ML та аналітики;
- популярність і завершеність.

Selenium – це набір інструментів, призначених для автоматизації WEB-браузерів на різних платформах. Selenium може автоматизувати безліч різноманітних браузерів на різних платформах, використовуючи різні мови програмування і інтегруючись з різними тестовими фреймворками. Також Selenium — це масштабний open source проєкт, а точніше, browser automation framework, у межах якого розробляється серія програмних продуктів для автоматизованого тестування, зазвичай використовуються для тестування WEB-додатків.

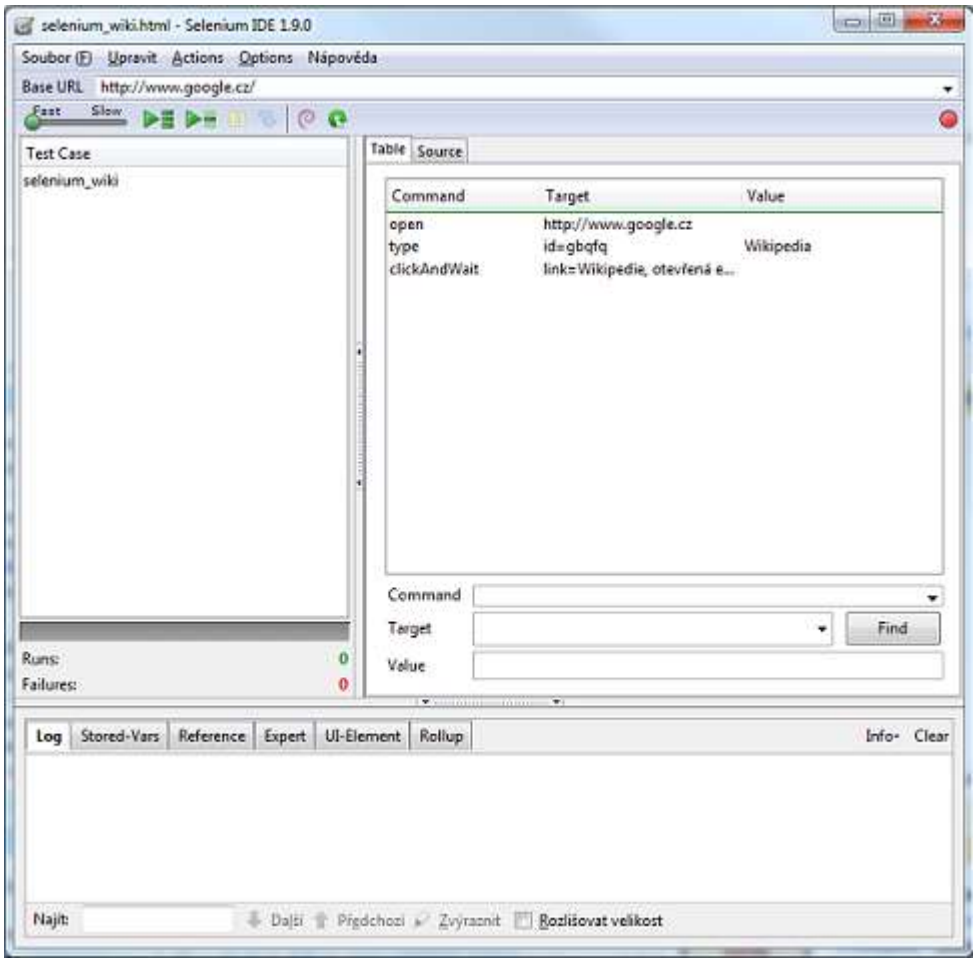


Рисунок 1.3 – Загальний вигляд інтерфейсного меню Selenium

«Selenium Core», ядро проєкту або «Selenium 1.0» в основі було зароджене з бібліотеки на JavaScript «JavaScriptTestRunner», яка була створена в 2004 році Джейсоном Хаггінсом у компанії ThoughtWorks, і призначалася для запуску тестів у браузері до сайту на Python. Згодом до розвитку проєкту приєдналися інші співробітники. А з 2007-го року Хаггінс з колегами працювали над Selenium вже у Google. Вийшло кілька версій, зокрема довгождана версія Selenium 3.0, окреслено якими користувачі побачать Selenium 4.0, Selenium 5.0.

Завдяки їх праці на сьогодні Selenium складається із багатьох бібліотек написаних на різних мовах програмування (хоча в основному на кросплатформенній Java).

До програмних продуктів Selenium входять: Selenium WebDriver [11], Selenium Server+Selenium Grid, Selenium IDE, Selenium RC.

Основною метою розробки системи автоматичного тестування WEB-додатків є надання розробникам можливості легко дізнатись про проблеми під час використання користувачами WEB-додатків на клієнтському рівні.

Вимоги до системи автоматичного тестування WEB-додатків [12]:

- реєстрація користувачів;
- можливість створити запит на автоматичне тестування за посиланням;
- можливість налаштувати відтерміновану/повторювану перевірку;
- наглядна демонстрація проблем з продуктивністю у WEB-додатку;
- збереження локальних результатів тестування.

Apache JMeter – це програмне забезпечення з відкритим вихідним кодом, 100% чиста програма Java, розроблена для завантаження тестової функціональної поведінки та вимірювання продуктивності. Спочатку він був розроблений для тестування WEB-додатків, але згодом розширився до інших тестових функцій [13].

Apache JMeter можна використовувати для тестування продуктивності як на статичних, так і на динамічних ресурсах, динамічних WEB-додатках. Його можна використовувати для імітації великого навантаження на сервер, групу серверів, мережу чи об’єкт, щоб перевірити його міцність або проаналізувати загальну продуктивність за різних типів навантаження, схематичне представлення зв’язку вузла контролера з робочими вузлами представлений на рисунку 1.4 [13].

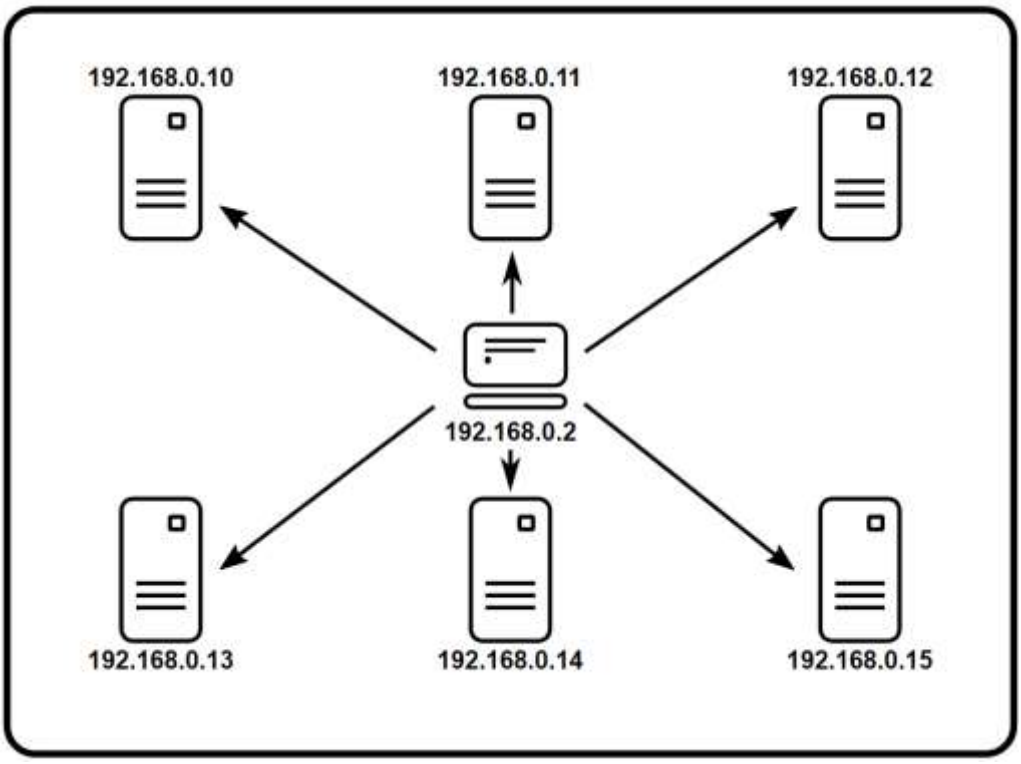


Рисунок 1.4 – Схема одновузлового контролера із кількома робочими вузлами

Функції Apache JMeter включають:

- можливість завантажувати та тестувати продуктивність багатьох різних типів програм/серверів/протоколів;
- веб протокол передачі даних – HTTP, HTTPS (Java, NodeJS, PHP, ASP.NET) [16];
- WEB-сервіси програмної розробки SOAP/REST [17];
- використання бази даних через JDBC [18];
- полегшений протокол доступу до директорії LDAP [19];
- проміжне програмне забезпечення, орієнтоване на повідомлення (MOM) через JMS;
- електронна пошта - SMTP(S), POP3(S) і IMAP(S);
- власні команди або сценарії оболонки;
- мережевий протокол TCP [20];
- об’єкти розробки на мові програмування Java;
- повнофункціональна тестова IDE, яка дозволяє швидко записувати

план тестування (з браузерів або нативних програм), створювати та налагоджувати;

− режим CLI (режим командного рядка (раніше називався Non GUI)/безголовий режим) [21] для завантаження тесту з будь-якої Java-сумісної ОС (Linux, Windows, Mac OSX);

− повний і готовий до представлення динамічний HTML-звіт;

− легка кореляція завдяки можливості отримувати дані з найбільш популярних форматів відповідей, HTML, JSON, XML або будь-якого текстового формату;

− повна мультипотокова структура дозволяє одночасну вибірку багатьма потоками та одночасну вибірку різних функцій окремими групами потоків;

− високорозширюване ядро;

− вставні пробовідбірники надають необмежені можливості тестування;

− збірники сценаріїв (сумісні з JSR223 мови, такі як Groovy та BeanShell);

− кілька статистичних даних про навантаження можна вибрати за допомогою підключених таймерів;

− плагіни аналізу даних і візуалізації забезпечують широке розширення та персоналізацію;

− функції можна використовувати для надання динамічних вхідних даних для тесту або забезпечення маніпулювання даними;

− легка безперервна інтеграція через сторонні бібліотеки з відкритим кодом для Maven, Gradle і Jenkins.

У спрощеному вигляді роботу середовища автоматизованого тестування JMeter можна розбити на окремі категорії, зв’язок яких схематично представлений на рисунку 1.5:

− Controller Node – система з графічним інтерфейсом користувача JMeter, який керує тестом;

− Worker Node – система, на якій працює сервер jmeter, який приймає команди з графічного інтерфейсу користувача та надсилає запити до цільової

системи (систем). Таких вузлів може бути кілька, в залежності від потреб користувача;

- Target – WEB-сервер, який планується протестувати.

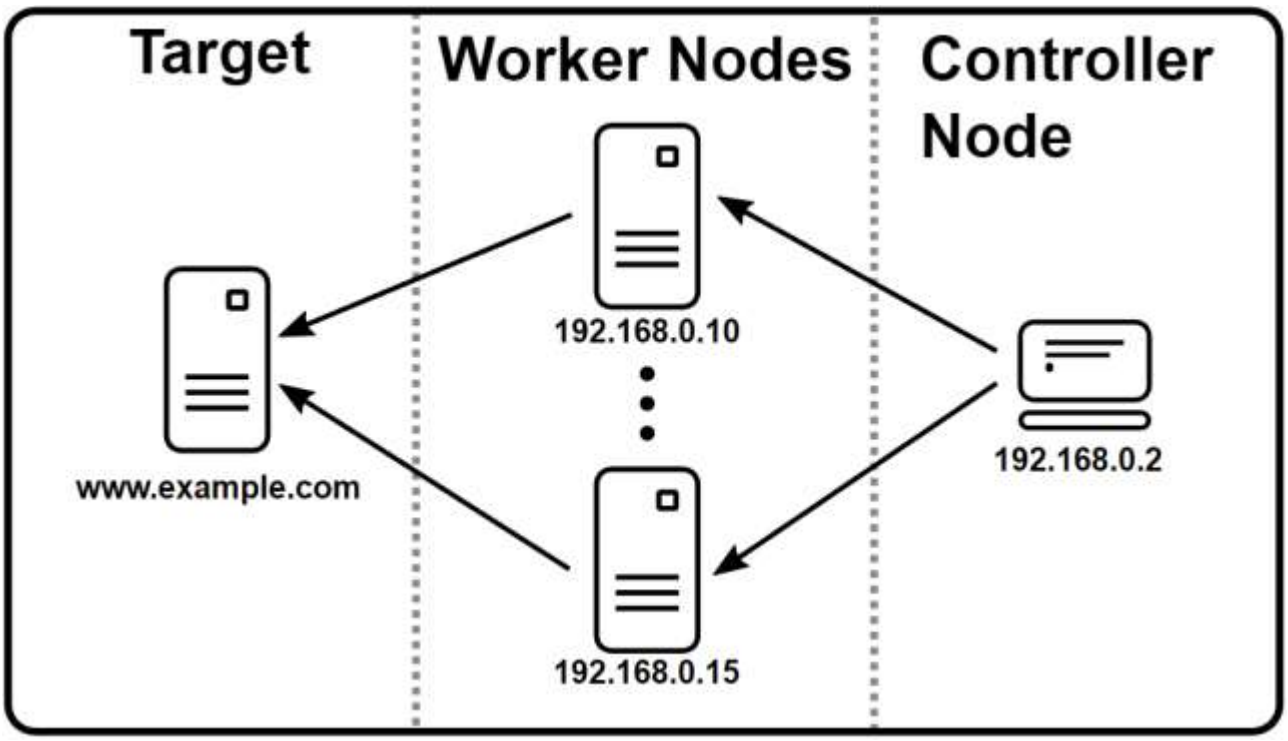


Рисунок 1.5 – Схема функціонування середовища автоматизованого тестування JMeter

Katalon Studio – це інструмент автоматизації з відкритим кодом, який підтримує як WEB-середовища, так і середовища розробки мобільних додатків.

Katalon Studio – це потужне комплексне рішення для автоматизації тестування API, тестування WEB-програм, мобільних пристроїв та настільних програм. Він також має багатий набір функцій для цих типів тестування та підтримує декілька платформ, включаючи Windows, macOS та Linux [23].

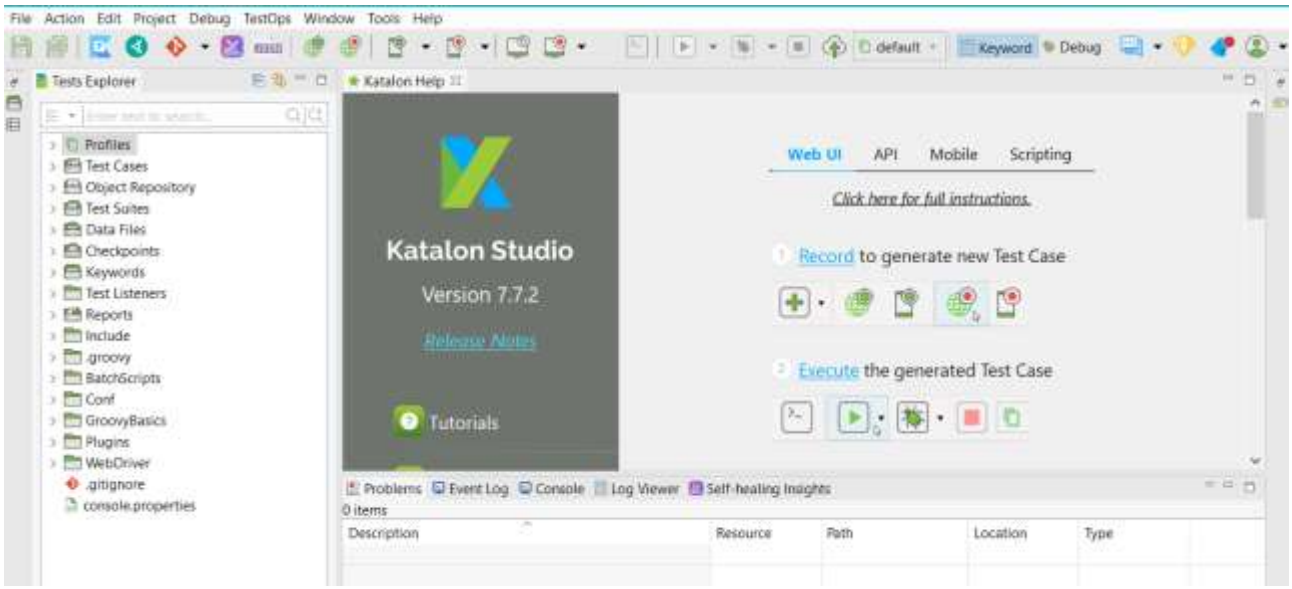


Рисунок 1.6 – Загальний вигляд інтерфейсного меню Katalon Studio

Основні особливості інструменту:

- мова програмування Groovy (з підтримкою Java);
- стиль програмування тестів: виклик методів класів (ківордів);
- підтримує BDD Cucumber для Behavior Driven Testing (у стилі When-And-Then);
- використовує движки Selenium та Appium;
- підтримує SOAP та RESTful для тестування API та сервісів;
- близько двухсот вбудованих ківордів;
- можливості тестування можуть бути розширені за допомогою створення ківордів, плагінів з Katalon Store, імпорту сторонніх бібліотек з файлів .jar;
- детальний перегляд звітів у Katalon TestOps.

Katalon Studio є одним із найкращих інструментів для тестування автоматизації, який працює зверху на Selenium та Appium, тим самим покращуючи ці фреймворки такими функціями, як об'єктний шпигун, зручна IDE, сховище об'єктів та плагін браузера. Він може бути інтегрований з безліччю інших інструментів, таких як JIRA, qTest, Kobiton, Git, Slack тощо.

Платформа виділяється своїми численними цілями та простотою

використання, враховуючи, що вона може створювати та використовувати повторно тестові сценарії інтерфейсу користувача, не вимагаючи жодного коду.

Інструмент використовує Groovy як мову сценаріїв та підтримує зовнішню бібліотеку Java. Katalon дозволяє повторно використовувати сценарії Selenium, написані на Java, і використовувати безпосередньо в інструменті. Він безперебійно працює з системами безперервної інтеграції, такими як Jenkins, Bamboo та TeamCity.

Уніфіковане функціональне тестування (рис. 1.7) Micro Focus (UFT), раніше відоме як QuickTest Professional (QTP) [24] – це програмне забезпечення, яке забезпечує автоматизацію функціональних і регресійних тестів для програмних додатків і середовищ [25]. UFT One (раніше відомий як UFT) – популярний комерційний інструмент для тестування WEB-програм, настільних ПК, мобільних пристроїв та додатків RPA. Він був розширений, включаючи хороший набір можливостей для тестування API. Підтримуючи кілька платформ для цільового тестованого додатка (AUT), UFT One пропонує зручний вибір для тестування AUT, який працює на настільних комп'ютерах, в Інтернеті та на мобільних пристроях.

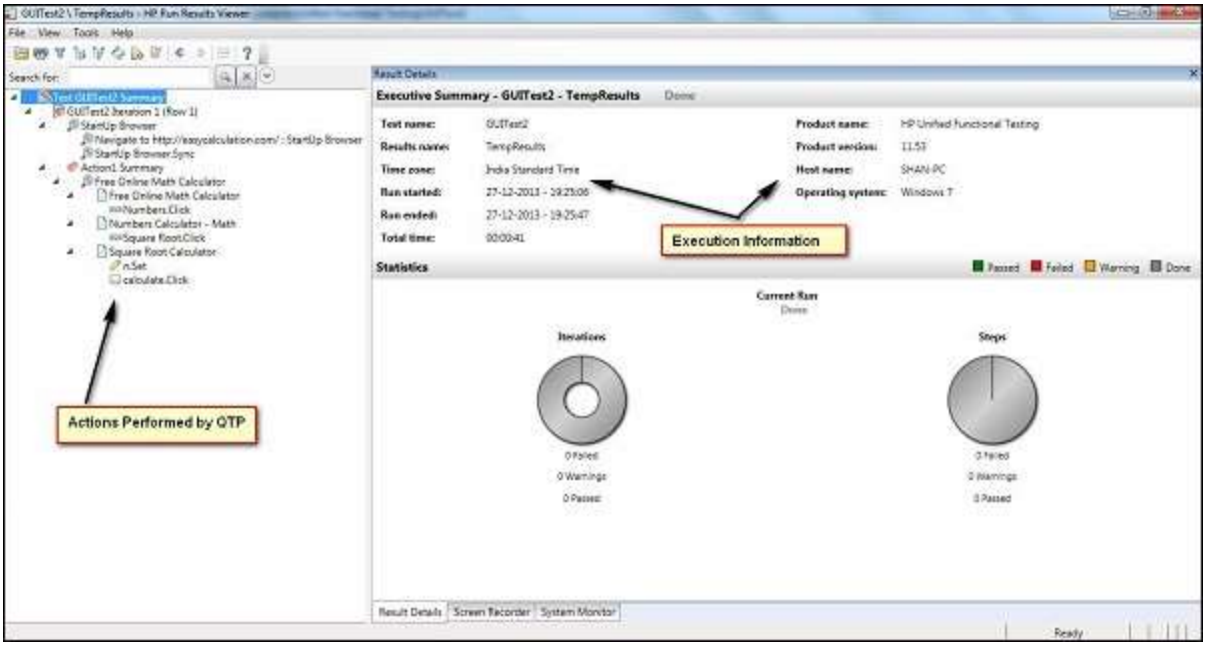


Рисунок 1.7 – Загальний вигляд інтерфейсного меню HPE Unified Functional Testing (UFT)

Він може автоматизувати інтернет-браузер, інтерфейс «Робочий стіл», SAP, Java, Oracle, Mobile та Visual Basic, серед інших програм. Список середовищ розробки, які він може автоматизувати, величезний, і його можна використовувати для різних видів тестування програмного забезпечення. Він виконує функціональне та регресійне тестування через користувальницький інтерфейс, такий як власний графічний інтерфейс або WEB-інтерфейс [26].

UFT використовує VBScript як мову сценаріїв. Інструмент тісно інтегрований з HP ALM (засіб управління тестами) та HP LoadRunner (інструмент тестування продуктивності).

1.3 Порівняльний аналіз характеристик серверних систем програмних засобів автоматизованого тестування WEB-додатків

Віддалений моніторинг – це технологія яка забезпечує дані про продуктивність сервера (разом з іншими показниками) з віддаленої системи; тобто тестований сервер передає дані через мережу тій частині інструменту тестування продуктивності, яка запускає програмне забезпечення для моніторингу. Велика перевага використання віддаленого моніторингу полягає в тому, що вам зазвичай не потрібно встановлювати будь-яке програмне забезпечення для моніторингу системи. Далі будуть наведені елементи статистики, які мають значення для збору результатів тестування продуктивності [23]:

– Стандартне відхилення і нормальний розподіл: іншим загальним і корисним показником є стандартне відхилення, яке відноситься до середньої дисперсії від розрахункового середнього значення. Віно базується на припущенні, що більшість даних про випадкові події в реальному житті мають нормальний розподіл. Чим вище стандартне відхилення, тим далі елементи даних, як правило, лежать від середнього. З точки зору тестування продуктивності, високе стандартне відхилення може свідчити про непостійний

досвід користувачів. Наприклад, у випадку використання може бути розрахований середній час відгуку 40 секунд, але стандартне відхилення 30 секунд. Це означало б, що кінцевий користувач має високі шанси відчути час відгуку від 25 до 55 секунд для тієї ж активності. Слід прагнути досягти невеликого стандартного відхилення.

Ймовірність виникнення збою роботи трапляється при неправильному виникненні запиту користувачем або забогато кешованих даних, допустимий час отримання відповіді залежить від швидкості обробки даних на сервері. Нижче наведем приклад ймовірності виникнення збою на рисунку 1.8, допустимий час отримання відповіді на рисунку 1.9 та степінь хешування паролів на рисунку 1.10.

Рисунок 1.8 – Діаграма ймовірності виникнення збою роботи програмного модуля серверної частини від вмісту хешованих даних на сервері

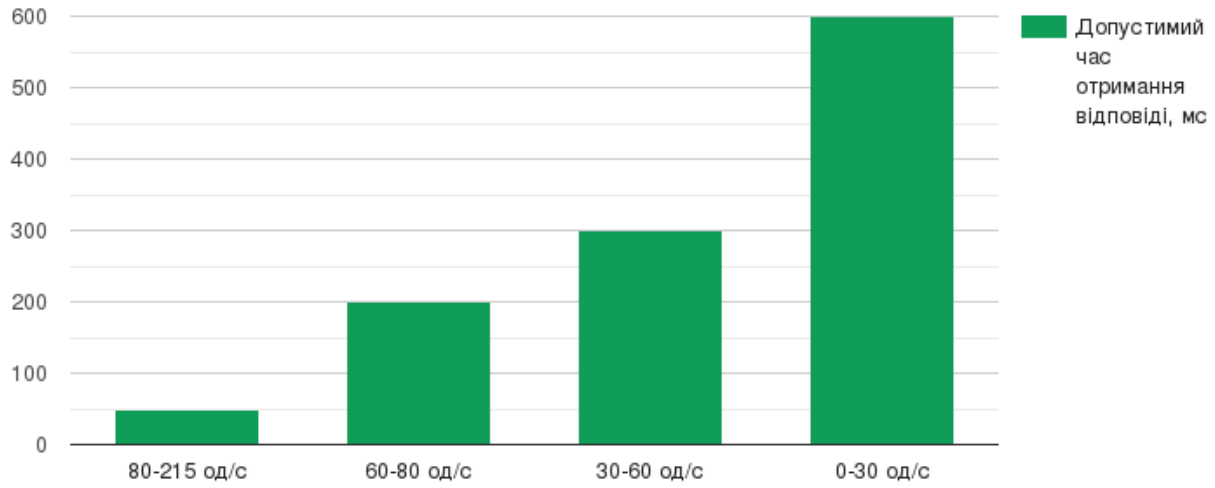


Рисунок 1.9 – Діаграма швидкості обробки даних на сервері та допустимий час отримання відповіді

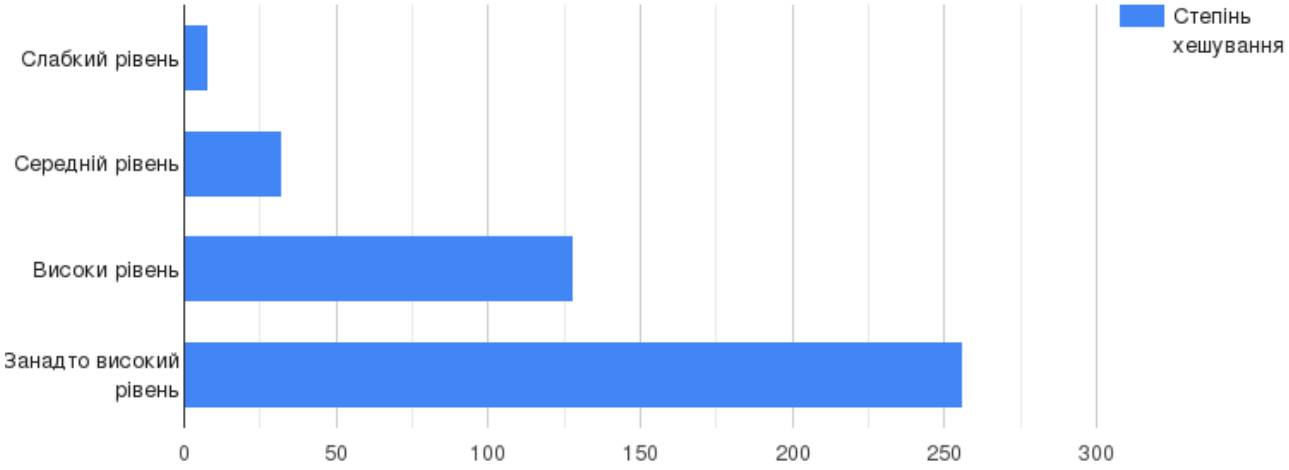


Рисунок 1.10 – Діаграма залежності рівня безпеки персональних даних в степеня хешування

– Пропускна здатність та потужність: поряд із часом відгуку 31 тестувальників продуктивності, як правило, найбільше цікавить, скільки даних або скільки випадків використання можна обробити одночасно. Можна сприймати це вимірювання як пропускну здатність, щоб підкреслити, наскільки швидко обробляється певна кількість випадків використання, або як здатність підкреслити, скільки випадків використання можна обробити певний період часу. Раптове зменшення пропускної здатності незмінно вказує на проблеми і

може збігатися з помилками, з якими стикаються один або кілька віртуальних користувачів. Ian Molyneaux пише що це часто трапляється, коли рівень WEB-сервера досягає своєї точки насиченості для вхідних запитів. Віртуальні користувачі починають зупинятися, чекаючи відповіді вебсерверів, що призводить до зменшення пропускної спроможності. Приклад графіка пропускної здатності та потужності зображений на рисунку 1.11.

Рисунок 1.11 – Діаграма відношення пропускної здатності сервера до потужності сервера

Розподіл часу відгуку заснований на нормальній моделі розподілу, це спосіб агрегування часу відгуку на основі кількості запитів, зібраних під час тесту продуктивності, у ряд груп або сегментів. В статті аналізу розподілу часу відгуку пишуть, що розуміючи розподіл часу відповіді на всі запити, ми можемо зосередитись на тих запитах, які мають найповільніший час відповіді [24].

Приклад графіка розподілу часу відгуку зображений на рисунку 1.12;

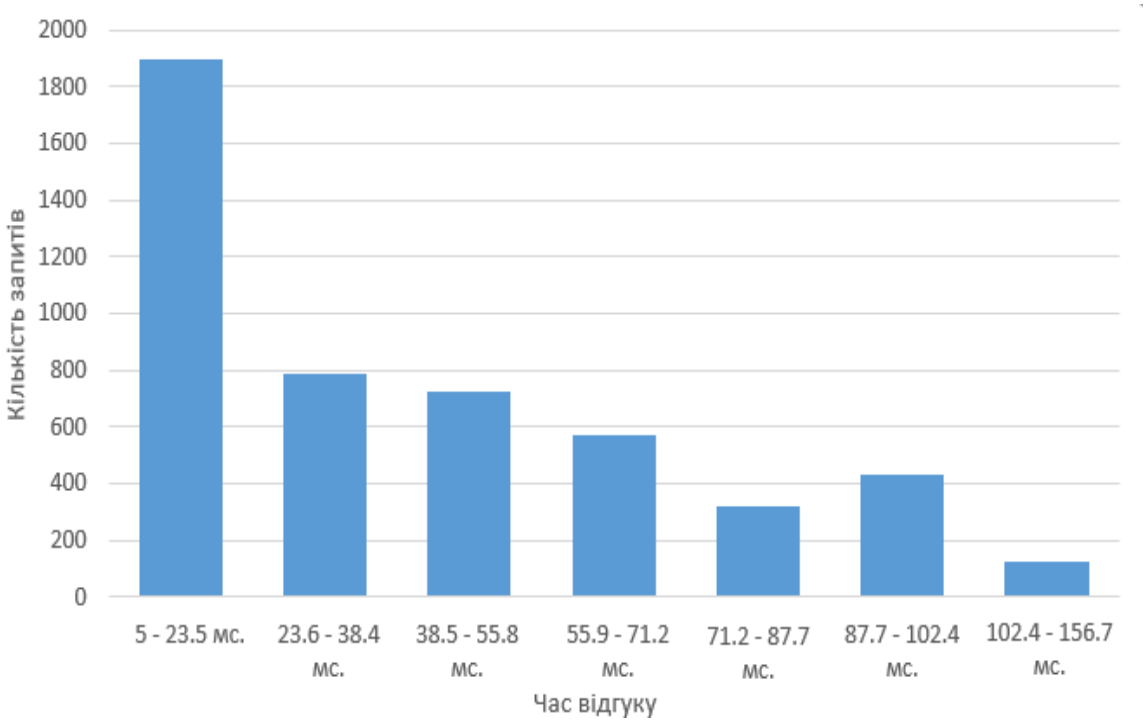


Рисунок 1.12 – Діаграма розподілу часу відгуку на всі запити

Зміцнення впевненості є важливою діяльністю при виконанні тесту на ефективність. Ніщо не забезпечує більшої впевненості в успішному виконанні тесту, як перегляд журналів виконання. Журнали виконання забезпечують прозорість виконання тесту та визначають, наскільки здоровим було виконання тесту. Спостереження журналів виконання надає статус відмови тестового сценарію під час виконання, тривалості тестових запусків, відмови рівня ОС, та збоїв рівня програми, якщо такі є [25].

Швидкість WEB-сайту залежить від різних факторів, таких як вміст на сторінках, браузер, географічне розташування доступу, пропускна здатність тощо. Часто можна зробити так, щоб вміст WEB-сторінки займав менше байт, не змінюючи зовнішній вигляд або функції сторінки. Зменшення кількості байтів, які повинен завантажити клієнт, пришвидшує завантаження сторінки [26].

1.4 Висновок до розділу 1

У сучасний час доступно багато автоматизованих інструментів тестування. Кожен інструмент цікавий, унікальний, має свої сильні та слабкі сторони. Різноманітність інструментів, що використовуються для автоматизованого тестування, ускладнює вибір інструментів, які найбільше потрібні для проєкту, і тестувальники часто отримують на фінальній стадії інструменти, які не відповідають вимогам проекту. Описано інструменти для автоматизованого тестування. Ми розглянули найважливіші критерії вибору інструменту. Ми порівняли три найпопулярніших інструменти автоматизації, оцінивши їх параметри, плюси і мінуси використання інструменту. Тому дуже важливо вибрати правильний інструмент для проєкту.

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ WEB-ДОДАТКІВ

2.1 Розробка підходів проведення статичного й динамічного тестування

До тестування програмного забезпечення існує багато підходів. Огляди, вичитка документації або перевірки називаються статичним тестуванням, тоді як виконання самого коду із заданим набором тестових випадків називається динамічним тестуванням. Статичне тестування часто є неявним, наприклад, корекція документації. Статичним аналізом програми також є перевірка засобами програмування/текстовими редакторами структури вихідного коду або компіляторами (попередніми компіляторами) синтаксису та потоку даних. Динамічне тестування відбувається під час запуску самої програми. Динамічне тестування може розпочатися до завершення програми для тестування окремих розділів коду та застосування їх до дискретних функцій або модулів [5, 6]. Статичне тестування передбачає верифікацію, тоді як динамічне тестування також передбачає валідацію [7].

Динамічне тестування – це методика, спрямована на перевірку функціоналу програми, під час виконання коду. Тобто, даний тип тестування передбачає експлуатацію програми і визначення того, чи відповідає результат виконання певного функціоналу очікуваному результату. Динамічний тип тестування складається з безпосереднього тестування програмного забезпечення в реальному часі, способом надання інформації на вхід і дослідження отриманого результату поведінки програми. Даний метод тестування допомагає команді перевірити різні критичні моменти програмного забезпечення. Якщо закрити очі на їхнє існування і ніяк не відреагувати на них,

це може певним чином позначитися на продуктивності, функціональній стороні і надійності програмного забезпечення.

Переваги методу динамічного тестування:

− у процесі тестування проводиться ретельне вивчення всього функціоналу програми, і, як результат, отримуємо високу якість перевірки;

− динамічне тестування здійснює перевірку програмного забезпечення з боку кінцевого користувача, що допомагає значно підвищити якість програмного забезпечення;

− фіксація комплексних багів (дефектів), які могли залишитися непоміченими;

− за допомогою спеціальних засобів динамічний тип тестування можна автоматизувати.

Недоліки методу динамічного тестування:

− налагодження та впровадження динамічного тестування потребує багато ресурсів та часу;

− динамічне тестування - дорогий процес;

− в основному, даний метод тестування застосовується під час розробки програмного забезпечення, а баги та дефекти виявляються в процесі життєвого циклу розробки.

Метод статичного тестування – це тип тестування програмного забезпечення, під час якого останнє перевіряється без запуску коду; є процесом або інструментом, спрямованим на виявлення можливих багів в програмному забезпечені. Крім цього, він знаходить і усуває помилки в різного роду супровідних документах, наприклад, в специфікації вимог до програмного забезпечення.

Статичне тестування ділиться на два типи:

− огляди – це тестування, спрямоване на виявлення дефектів в документації (вимоги, дизайнерське оформлення, тестові випадки тощо)

− статичний аналіз – це аналіз на наявність прогалин у структурі, здатних

привести до некоректного функціонування програмного забезпечення. Також застосовується для пошуку коду, що потенційно містить вразливості SAST (Static Application Security Testing) [8]. Під час статичного аналізу зазвичай перевіряється якість написаного розробниками коду. Для його проведення використовуються різні інструменти, такі як компілятори, лінкери, спеціальні утиліти.

За допомогою статичного аналізу можна виявити наступні дефекти:

- змінні, які не використовуються;
- не ініційовані змінні;
- некоректний синтаксис;
- переповнення буфера;
- нескінченні цикли;
- незмінний параметр, який передається у функцію;
- дублікати коду.

Переваги методу статичного тестування:

- виконується до розгортання та виконання коду;
- знаходить помилки на перших етапах розробки програмного забезпечення, що сприяє значному зменшенню вартості їх виправлення;
- відгуки, отримані в процесі даного тестування допомагають удосконалити процес розробки програмного забезпечення та зменшити вірогідність виникнення схожих багів;
- надає високий рівень інформативності щодо проблем програмного забезпечення;
- сприяє ліпшому обміну інформацією між співробітниками;
- виправлення багів та дефектів на початкових етапах потребує значно меншої кількості зусиль, що дозволяє прискорити розробку та зменшити затрати на неї.

Недоліки методу статичного тестування:

- часто потребує сторонніх засобів тестування, їх налагодження та

використання в ручному режимі, тобто довший час тестування;

− приховує певні вразливості, які стосуються середовища виконання програмного забезпечення;

− не дозволяє виявити дефекти в самій логіці програмного забезпечення.

2.2 Математична модель динамічного тестування методом «чорної скриньки»

Для проведення експерименту необхідно мати можливість впливати на поведінку «чорної скриньки». Всі способи такого впливу позначено літерою x і називається факторами які є також входами «чорної скриньки». Схема чорної скриньки зображена на рисунку 2.1.

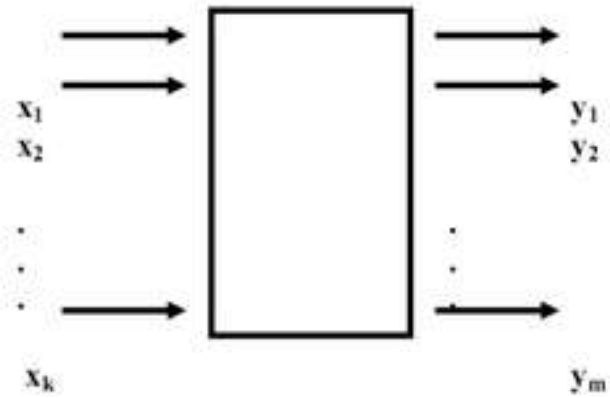


Рисунок 2.1 – Принципова схема моделі тестування «чорна скринька»

При вирішенні задачі використовують математичні моделі дослідження. Під математичною моделлю є рівняння, що зв'язує параметр оптимізації з чинниками.

Це рівняння в загальному вигляді можна записати так: $y = \varphi(x_1, x_2, \ldots, x_K)$ – функція відгуку.

Кожен фактор може приймати в досліді одне з декількох значень. Ці значення називаються рівнями. Для полегшення побудови «чорної скриньки» і експерименту фактор повинен мати певне число дискретних рівнів. Фіксований набір рівнів факторів визначає одне з можливих станів «чорної скриньки».

Одночасно це є умовою проведення одного з можливих дослідів. Якщо перебрати всі можливі набори станів, то виходить безліч різних станів «чорної скриньки». Це буде число можливих різних дослідів.

Число можливих дослідів визначають за виразом: $N = P^K$, де N - число дослідів; p - число рівнів; K - число факторів.

Щоб вивести основні особливості задачі оцінки чорної скриньки, було розглянуто простий приклад. Задача полягає в оцінці невідомої функції $g_0(x), -1 \leq x \leq 1$. Спостереження - це вимірювання шуму $y(k)$ в точках x_k:

$$y(k) = g_{0(x_k)} + e(k)\,. \tag{2.1}$$

Так чи інакше–повинно вирішитись «де шукати» параметр g. Через інформацію про те, що g є многочленом третього порядку. Це призводить сірої скриньки моделі:

$$g(x,\theta) = \theta_1 + \theta_2 x + \theta_3 x^2 + \cdots + \theta_n x^{n-1}, \tag{2.2}$$

де $n = 4$, і оцінка вектору θ параметра зі спостережень $y(k)$.

Є кілька альтернатив з чорною скринькою:

Можливе використання раціональних наближень

$$g(x,\theta) = \frac{\theta_1+\theta_2 x+\theta_3 x^2+\cdots+\theta_n x^{n-1}}{1+\theta_{n+1} x+\theta_{n+2} x^2+\cdots+\theta_{n+m-1} x^{xn-1}}\,, \tag{2.3}$$

або розширення ряду Фур'є:

$$g(x,\theta) = \theta_0 + \sum_{l=1}^{n} \theta_{2l-1}\cos(l\pi x) + \theta_{2l}\sin(l\pi x)\,. \tag{2.4}$$

Як варіант, можливо апроксимувати функцію за допомогою відрядних постійних функцій, як це проілюстровано на рисунку 2.2.

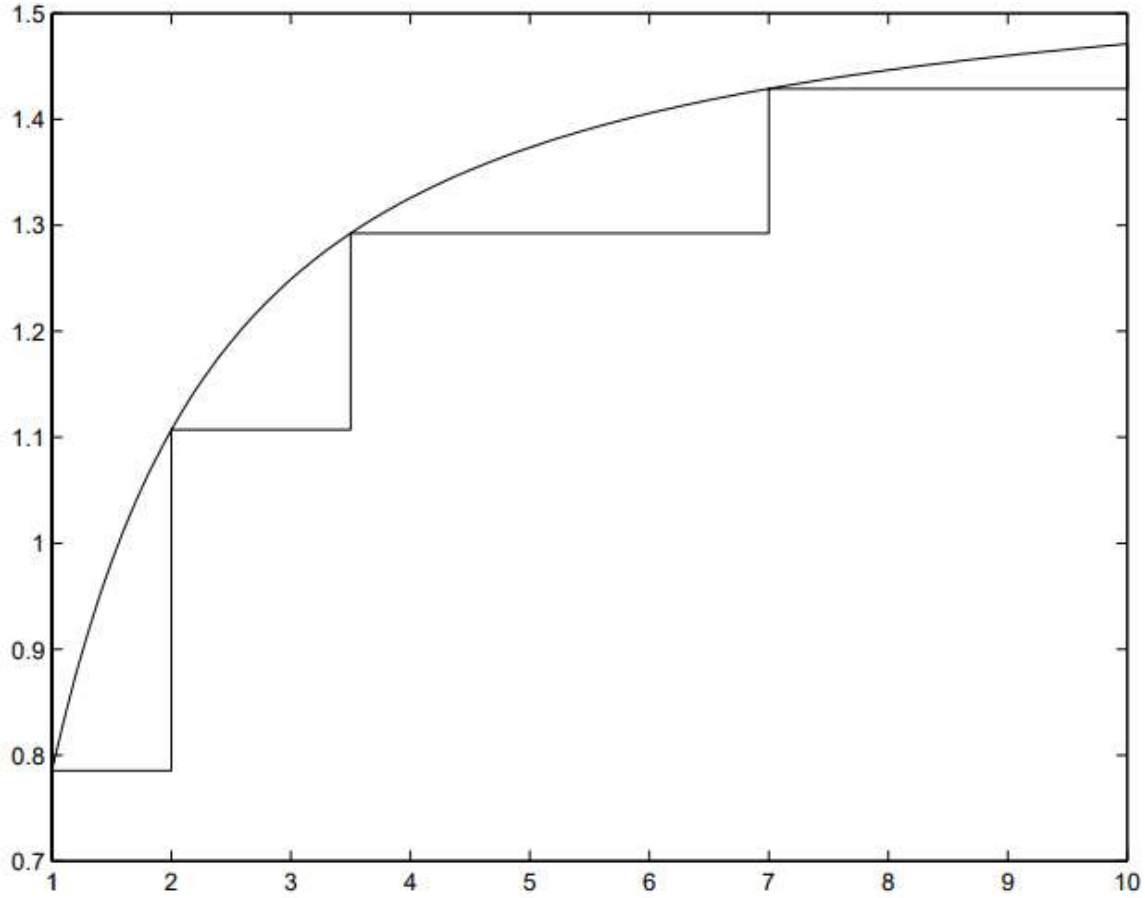


Рисунок 2.2 – Діаграма частково-постійної апроксимації ряду Фур’є

Основні етапи моделювання чорної скриньки наступні:

1. Взяття "типу" класу структури моделі. (Наприклад, у випадку, наведеному вище, перетворення Фур’є, раціональна функція або відрядна константа).

2. Визначення «розміру» даної моделі (тобто кількість параметрів, n). Це буде відповідати тому, наскільки «відмінно» буде наближення.

3. Використання даних спостережень як для оцінки числових значень параметрів, так і для вибору відповідного значення n. Позначається модельна структура для спостереження:

$$\hat{y}(k|\theta) = g(x_k, \theta). \tag{2.5}$$

Вона полягає в інтерпретації таким чином, що $\hat{y}(k|\theta)$ є прогнозованим значенням спостереження $y(k)$, припускаючи, що функція може бути описана параметром vector θ.

Також в цілому позначено всі вимірювання, доступні до часу N, по Z^N (2.6).

Принципи і алгоритми припасування моделей до даних, а також загальні властивості оцінюваних моделей - все це модель-структура незалежна і однаково добре застосовна до, скажімо, лінійним моделям ARMAX і моделям нейронних мереж, про які піде мова далі в даній роботі.

Основні найменші квадрати, такі як підхід, є природним підходом, навіть коли предикатор $\hat{y}(t|\theta)$ є більш загальною функцією θ:

$$\widehat{\theta_N} = arg \min_{\theta} V_N(\theta, Z^N), \tag{2.7}$$

де

$$V_N(\theta; Z^N) = \frac{1}{N} \sum_{t=1}^{N} \|y(t) - \hat{y}(t|\theta)\|. \tag{2.8}$$

Також використовуються наступні позначення для невідповідності між вимірюванням і прогнозованою вартістю:

$$\varepsilon(t;\ \theta) =\ y(t) - \hat{y}(t;|\theta). \tag{2.9}$$

Ця процедура є природною і прагматичною – людина може думати про неї як про «криву», що відповідає між $y(t)$ і $\hat{y}(t|\theta)$. Процедура також має кілька статистичних та інформаційних теоретичних інтерпретацій. Найголовніше, якщо джерелом шуму в системі передбачається гауссова послідовність незалежних випадкових величин $\{e(t)\}$ тоді (2.7) стає оцінкою максимальної ймовірності (МЛЕ).

Якщо $\hat{y}(k|\theta)$ — лінійна функція задачі θ мінімізації (2.7) легко вирішується. У більш загальних випадках мінімізацію доведеться здійснювати шляхом ітераційного (локального) пошуку по мінімуму:

$$\hat{\theta}_N^{i+1} = \hat{\theta}_N^i + \mu f\left(Z^N, \hat{\theta}_N^i\right), \tag{2.10}$$

де f – параметр зв'язаний із градієнтом V_N , як напрямок Гаусса-Ньютона [24].

Також досить корисна робота зі зміненим критерієм

$$W_N(\theta; Z^N) = V_N(Z^N) + \delta\left|\left|\theta\right|\right|^2, \tag{2.11}$$

з V_N визначеним (2.8). Це відомо як регуляризація. Можна відзначити, що зупинка ітерацій (i) в (2.10) до досягнення мінімуму має такий самий ефект, як і регуляризація [25].

Модель $g(x;\ \hat{\theta}_N)$ буде помилкова двома способами:

По-перше, буде розбіжність між граничною моделлю $g(x;\ \theta^*)$ та істинною функцією $g_0(x)$, оскільки наші структурні припущення не є правильними, наприклад, функція не є відрядною константою. Ця помилка називається помилкою зміщення.

По-друге, буде розбіжність між фактичною оцінкою $\hat{\theta}_N$ і граничним значенням. Це пов'язано з шумокорумпованими вимірами (термін $e(k)$ в (2.1)).

Повинно бути зрозуміло, що між цими аспектами існує компроміс: щоб отримати меншу помилку упередження, людина повинна, наприклад, використовувати більш тонку сітку для штучного наближення. Це, в свою чергу, означало б, що менше спостережуваних даних може бути використано для оцінки кожного рівня часткового наближення функції, що призводить до великої дисперсії для цих оцінок.

Таким чином, боротьба з компромісом між упередженнями та дисперсією лежить в основі вибору структури моделі чорної скриньки. Деякі досить

загальні вирази для очікуваної моделі підходять, які не залежать від структури моделі, можуть бути розроблені.

θ^* мінімізує очікуване значення критерію (2.8). Позначення $dim\,\theta$ означає кількість передбачуваних параметрів. Результат також передбачає, що структура моделі успішна в тому сенсі, що $\varepsilon(t)$ - це приблизно білий шум. Досить важливо відзначити, що число $dim\,\theta$ (2.18) буде змінено на число власних значень $\bar{V}''(\theta)$ (Гессіан з $\bar{V}$), які більші, ніж у випадку регуляризованої функції втрат (2.11) зводиться до мінімуму для визначення кошторису. Незважаючи на застереження щодо формальної дійсності (2.18), він несе найважливіший концептуальний меседж: Якщо модель оцінюється на наборі даних з тими ж властивостями, що і дані оцінки, то відповідність не буде залежати від властивостей даних, і вона буде залежати від структури моделі тільки з точки зору кількості використовуваних параметрів і найкращої відповідності, пропонованої в рамках структури. Чим більше параметрів використовується структурою, тим вище термін дисперсії, але при цьому нижче прилягання $\bar{V}(\theta^*)$. Компроміс, таким чином, полягає в тому, щоб збільшити ефективну кількість параметрів тільки до того моменту, коли поліпшення посадки за параметром перевищує $\bar{V}(\theta^*)/N$. Цього можна досягти, оцінивши F_N в шляхом оцінки функції втрат при $\hat{\theta}_N$ для набору даних перевірки. Це також може бути досягнуто за допомогою процедур Акайке [26], вираз можна переписати наступним чином. Нехай $g_0(t)$ позначає «істинне» на крок попереду передбачення y(t), а нехай:

$$W(\theta) = E|g_0(t)\hat{y}(t|\theta)^2, \tag{2.19}$$

і нехай

$$l = E|y(t) - g_0(t)|^2. \tag{2.20}$$

Мінімізувати (2.20) полягає в розбиті спостережувальних даних на оціночному наборі і наборі валідацій.

Оціночні моделі, використовуючи набір даних оцінки, для ряду різних розмірів вектора параметра. Обчислити значення критерію для кожної з цих моделей, використовуючи дані перевірки. Вибрати в якості остаточної моделі ту, яка мінімізує придатність даних для перевірки. Альтернативою є пропозиція великої кількості параметрів у фіксованій структурі моделі і використання регуляризованого критерію (2.11). Оцінити кілька моделей, так як повертає «ручку» з більших значень в сторону нуля і оцінення отриманих моделів.

Лінійна система унікально визначена та описана її частотною функцією $G(e^{i\omega})$ (тобто перетворення Фур'є своєї імпульсної реакції). Таким чином, можливо пов'язати оцінку лінійних систем безпосередньо з задачею оцінки функції (2.1), візьмемо $x_k = e^{i\omega k}$ і дозволити g бути комплексно-вартісним. Оскільки спостереження за входом-виходом безпосередньо беруться або перетворюються на частотну область (y тут були б невизначеними спостереженнями частотної характеристики на певних частотах), ми маємо пряму задачу оцінки функції. Параметризація $G(e^{i\omega})$ може бути зроблена за допомогою штучних постійних функцій, що тісно пов'язано зі стандартним спектральним аналізом [24, 27]. В іншому випадку частіше параметризувати їх як раціональні функції в (2.3). Пряме прилягання раціональних частотних функцій до даних спостережуваної частотної області розглядається комплексно в [24, 28].

Якщо спостереження y, які будуть використовуватися для прилягання моделі, є вхідними даними в часовій області, це діє наступним чином: Припустимо, що дані були згенеровані відповідно до

$$y(t) = G(q,\theta)u(t) + H(q,\theta)e(t), \qquad (2.22)$$

де e – білий шум (непередбачуваний), q – оператор прямого зсуву, а H – монічний (тобто його розширення в q_1 починається з матриці ідентичності).

Також припускається, що G містить затримку. Переписується (2.22) як:

$$y(t) = [I - H^{-1}(q,\theta)]y(t) + H^{-1}(q,\theta)G(q,\theta)u(t) + e(t).$$

Перший доданок в RHS містить тільки $y(t-k);\ k \leq 1$ так природний предиктор $y(t)$, заснований на минулих даних, буде заданий:

$$\hat{y}(t|\theta) = W_y(q,\theta)y(t) + W_u(q,\theta)u(t), \quad (2.23)$$

$$W_y(q;\theta) = [I - H^{-1}(q,\theta)], \quad (2.24)$$

$$W_u(q,\theta) = \text{H}^{-1}(q,\theta)G(q,\theta). \quad (2.25)$$

Такі значення і фільтри $H^{-1}G$ і H^{-1} були стабільними.

Лінійні моделі вводу-виводу чорної скриньки. У випадку з чорним ящиком дуже природним підходом є опис G і H in (2.22) як раціональних передаточних функцій в операторі зсуву (затримки) з невідомим чисельником і знаменником многочленом asn в (2.3). Тоді було б:

$$G(q,\theta) = \frac{B(q)}{F(q)} = \frac{b_1 q^{-nk-1} + \cdots + b_{nb} q^{-nk-nb+1}}{1 + f_1 q^{-1} + \cdots + f_{nf} q^{-nf}}. \quad (2.26)$$

Тоді $\eta(t) = G(q,\theta)u(t)$ (2.27) - скорочене позначення для відношення:

$$\eta(t) + f_1\eta(t-1) + \cdots + f_{nf}\eta(t-nf) =$$
$$b_1 u(t-nk) + \cdots + b_{nb} u\big(t - (nb + nk - 1)\big). \quad (2.28)$$

Тут спостерігається тимчасова затримка зразків nk. Точно так само функцію передачі обурення можна записати:

$$H(q,\theta) = \frac{C(q)}{D(q)} = \frac{1 + c_1 q^{-1} + \cdots + c_{nc} q^{-nc}}{1 + d_1 q^{-1} + \cdots + d_{nd} q^{-nd}}. \quad (2.29)$$

Вектор параметра, таким чином, містить коефіцієнти b_i ; c_i; d_i ; і f_i функцій передачі. Ця модель, таким чином, описується п'ятьма структурними параметрами: nb; nc; nd; nf ; і nk і відомий як модель Box-Jenkins (BJ). Важливим окремим випадком є випадки, коли властивості сигналів обурення не моделюються, а шумова модель $H(q)$ вибирається як $H(q)$ 1; тобто $nc = nd = 0$. Цей особливий випадок відомий як модель помилки виводу (OE), оскільки джерело шуму e$(t) = v(t)$ тоді буде різницею (похибкою) між фактичним виходом і безшумним виходом. Поширеним варіантом є використання одного і того ж еномінатора для G і H:

$$F(q) = D(q) = A(q) = 1 + a_1 q^{-1} + \cdots + a_{na} q^{-na} . \qquad (2.30)$$

Множення обох сторін (2.26)-(2.29) на $A(q)$ потім дає:

$$A(q)y(t) = B(q)u(t) + C(q)e(t) . \qquad (2.31)$$

Ця модель відома як модель ARMAX [30]. Назва походить від того факту, що $A(q)y(t)$ представляє авторегресію, а $C(q)e(t)$ ковзне середнє білого шуму, тоді як $B(q)u(t)$. Окремий випадок $C(q) = 1$ дає багато використовувану модель ARX. На рисунку 2.3 зображені різні моделі.

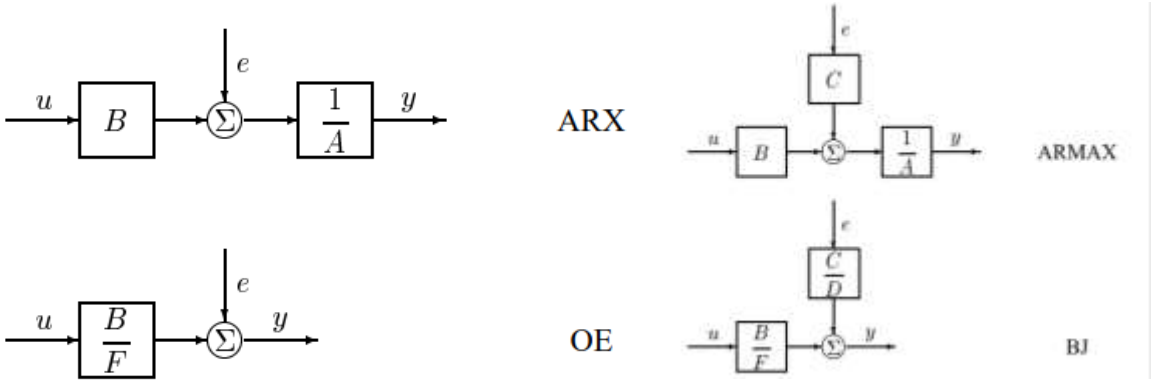


Рисунок 2.3 – Різні моделі чорної скриньки

Взявши за основу модель оцінки надійності програмного забезпечення Гокаля [2] , надійність цілої системи обчислюється за формулою:

$$R = \prod_{i=1}^{n} R_i. \tag{2.32}$$

У свою чергу надійність кожної компоненти з використанням Марковських процесів вищого порядку обчислюється за формулою:

$$R_l = e^{-\int_0^{V_{ij\dots k} t_{ij\dots kl}} \lambda_l(t)dt,} \quad . \tag{2.33}$$

Для знаходження $V_{ij\dots kl}$ – очікуваної кількості відвідувань компоненти і в залежності від виконання попередніх N компонент, потрібно розв'язати систему лінійних рівнянь:

$$V_{ij\dots kl} = q_{ij\dots kl} + \sum_{i=1}^{n-1} V_{ij\dots kl} P_{ij\dots kl}. \tag{2.34}$$

Окрім того з протоколу виконання програмного забезпечення потрібно обчислити такі числові вхідні параметри: $p_{ij\dots kl}$ – ймовірність переходу в компоненту l в залежності від виконання попередніх N компонент; $q_{ij\dots kl}$ – початковий ймовірнісний вектор; $t_{ij\dots kl}$ – час виконання компоненти l у залежності від виконання попередніх N компонент.

Для знаходження інтенсивності відмов кожної компоненти можна скористатись моделями «чорної скриньки», побудованих на основі результатів модульного тестування: модель Гоеля-Окумото, Муси, S-подібну або модель з динамічним показником складності проекту [11]. Отримавши числові значення усіх параметрів моделі, можна обчислити надійність кожної компоненти за формулою (2.33) та значення надійності комп'ютерної програми у цілому (2.32).

2.3 Розробка алгоритму побудови тестових сценаріїв методом «чорної» скриньки для автоматизованого тестування WEB-додатків

Вважатимемо, що кожен сценарій тестування складається з декількох кроків $T^k = \{ T_k^0, T_k^1, \dots, T_k^n \}$, де кожен крок сценарію $T_j^k = (S_j^k, C_j^k)$ визначається парою: методом S_j^k та множиною змінних C_j^k з відповідними класами еквівалентності, що точно змінюються в цьому методі. Змінні набувають своїх значень на кожному кроці тесту внаслідок введення даних користувачем, зчитуванням з бази даних, через що можуть виникнути помилки під час виконання даного сценарію тестування.

Варто зауважити, що у відповідності до реального процесу тестування, набори тестів будуються ітераційно, крок за кроком виконуючи їх на програмному забезпеченні та враховуючи виявлені помилки під час побудови наступного набору тестів.

Введемо деякі позначення: $Er^j(S^i)$ – число тестів, під час яких відбулась відмова, і які включають метод S^i на j-тій ітерації. $M(S^i)$ – множина номерів усіх методів, що мають переходи із компоненти S^i; $P(S^i)$ – множина всіх методів, у які можна передати контроль з методу S^i; N – число всіх вершин графу поведінки програмної системи; TS – набір усіх сценаріїв тестування; $coverage(S^i)$ – кількість входжень в метод S^i .

Алгоритм автоматизованої побудови тестових сценаріїв який зображений на рисунку 2.4 складається із двох основних частин: початкова генерація тестів без інформації про відмови та решти K наборів тестів, що її включають.

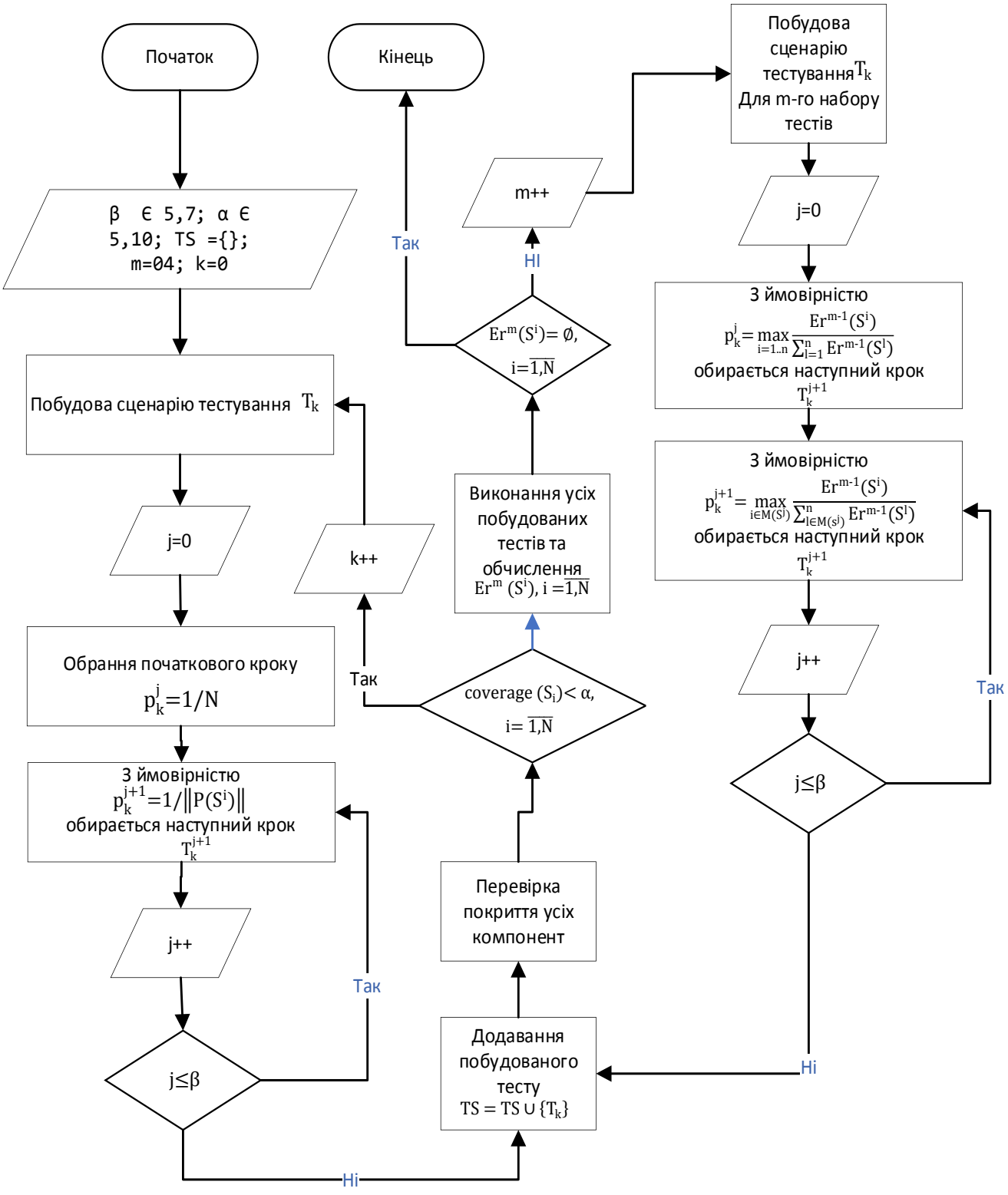


Рисунок 2.4 – Загальний вигляд схеми алгоритму автоматизованої побудови тестових сценаріїв

Побудовані послідовності методів виконуються при різних вхідних даних таким чином, щоб кожний вхідний параметр набував принаймні одного значення з кожного класу еквівалентності. Класи еквівалентності у методах чорної скриньки будуються не з аналізу коду, а відповідно до досвіду тестувальника. При цьому вхідні дані можуть і не набути усіх можливих значень і деякі відмови будуть пропущені, що є недоліком даної стратегії тестування.

Наприклад, для того, щоб повністю покрити код ПЗ, необхідно для кожного метода S^i здійснити $\|v_1\| \cdot \|v_2\| \cdots \|v_n\|$ перевірок, де $\|v_j\|$ – кількість класів еквівалентності змінної $v_j \in S^i (j = \overline{1..n})$. Але через обмеження часових та людських ресурсів не можливо зробити таку велику кількість перевірок. Окрім того не всі перевірки відповідають реальній логіці програми (певних значень змінних не можливо досягнути під час використання програми), тому існує задача покриття коду тестами так, щоб повністю перебрати усі можливі набори змінних, допустимих внутрішньою логікою, або, якщо таких випадків дуже багато, то перебрати усі кортежі класів еквівалентності змінних довжиною до деякого степеня m, який визначається співвідношенням вимог до якості ПЗ та затратами на процес тестування. Стандартно ($m = 1$) обмежуються перебором усіх класів еквівалентності для кожної змінної у методі $S^i - \|v_1\| + \|v_2\| + \cdots + \|v_n\|$. Для $m = 2$ необхідно перевірити всі пари наборів класів еквівалентності у методі $S^i - \sum_{k,j=1}^{n} \|v_k\| \cdot \|v_j\|$.

2.4 Висновок до розділу 2

У даному розділі магістерської кваліфікаційної роботи було розроблено і досліджено математичну модель «чорної скриньки» для тестування WEB-додатків.

Були побудовані набори тестів на кожній ітерації методом «чорної скриньки», які можуть бути використані під час тестування, що у свою чергу дозволяє пришвидшити даний процес та зменшити фінансові та людські витрати. Окрім того, така генерація сценаріїв забезпечує рівномірне покриття коду, що покращує ефективність тестування, та відповідно підвищує якість ПЗ. Також завдяки рівномірному покриттю коду та використанню класів еквівалентності, ймовірність виявлення відмови зростає під час тестування програми, що в результаті збільшує надійність ПЗ.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ WEB-ДОДАТКІВ

3.1 Обґрунтування вибору мови програмування

Перед початком розробки інформаційної технології автоматизованого тестування WEB-додатків потрібно обрати мову програмування, на якій вона буде написана.

Python [37] - популярна мова програмування, створена Гідо ван Россумом у 1991 році, яка використовується у багатьох сферах: WEB-розробка (на стороні сервера); розробка програмного забезпечення; математика; системний сценарій. Також Python має такі можливості: створення WEB-додатків на сервері; створення робочих процесів; підключення до систем баз даних; можна читати та змінювати файли; обробка великих даних та виконання складної математики; швидка розробка програмного забезпечення, готового до виробництва.

Python працює на різних платформах (Windows, Mac, Linux, Raspberry Pi [38] тощо). У Python є простий синтаксис, подібний до англійської мови. У Python є синтаксис, який дозволяє розробникам писати програми з меншою кількістю рядків, ніж деякі інші мови програмування. Python працює в системі інтерпретатора, тобто код може бути виконаний, як тільки він буде записаний. Це означає, що прототипування може бути дуже швидким.

Найновішою основною версією Python є Python 3.0, яка використовується при написанні програми. Однак Python 2, хоча не оновлюється нічим іншим, окрім оновлень безпеки, все ще залишається досить популярним.

Python може буде записаний у текстовому редакторі. Можна писати Python в інтегрованому середовищі розробки, таких як Thonny [39], Pycharm,

Netbeans або Eclipse, які особливо корисні при керуванні більшими колекціями файлів Python.

Синтаксис Python трохи схожий на інші мови програмування, але все одно має свої переваги порівняно з іншими мовами: Python був розроблений для читабельності і має деякі схожість з англійською мовою з впливом математики; Python використовує нові рядки для завершення команди, на відміну від інших мов програмування, які часто використовують крапки з комою або круглими дужками; для визначення області застосування Python покладається на відступ, використовуючи пробіли, наприклад, область циклів, функції та класи. Інші мови програмування часто використовують для цієї мети фігурні дужки.

Python порівняно простий, тому його легко вивчити, оскільки він вимагає унікального синтаксису, який зосереджений на читанні. Розробники можуть читати та перекладати код Python набагато простіше, ніж інші мови. У свою чергу, це зменшує витрати на підтримку та розробку програми, оскільки дозволяє командам працювати спільно без значних бар'єрів у мові та досвіді.

Крім того, Python підтримує використання модулів і пакетів, а це означає, що програми можуть бути спроектовані в модульному стилі, а код можна використовувати повторно в різних проектах. Розробивши потрібний вам модуль або пакет, його можна масштабувати для використання в інших проектах, і легко імпортувати або експортувати ці модулі.

Однією з найбільш перспективних переваг Python є те, що і стандартна бібліотека, і інтерпретатор доступні безкоштовно, як у двійковій, так і у вихідній формі. Ексклюзивності також немає, оскільки Python та всі необхідні інструменти доступні на всіх основних платформах. Тому це привабливий варіант для розробників, які не хочуть турбуватися про оплату високих витрат на розробку.

Python – потужна мова програмування, яка підтримує багато парадигм, оптимізована під забезпечення високої продуктивності програмістів, читабельності коду і якості програмного забезпечення [2].

Переваги: простий у написанні; низький поріг входженні; високорівнева; лаконічний (потрібно менше кількість рядків коду для написання більшості речей ніж на інших мовах програмування); велика кількість бібліотек та фреймворків () для роботи з ШІ, МН та ГН; може бути використаний для розробки майже будь-якого продукту (Web, Desktop, etc.). Недоліки: низька швидкодія; погана захищеність.

R – мова програмування і програмне середовище для статистичних обчислень, аналізу та зображення даних в графічному вигляді [21].

Переваги: високорівнева; велика кількість бібліотек для ШІ, МН;

Недоліки: складний у написанні через велику гнучкість та відсутність загальноприйнятих норм; великий поріг входження; низька швидкодія; багатослівна у порівнянні з Python; погана захищеність; вузька спеціалізованість;

C++ – мова програмування високого рівня з підтримкою декількох парадигм програмування: об'єктно-орієнтованої, узагальненої та процедурної. Створена Б'ярном Страуструпом [22].

Переваги: висока швидкодія; хороша захищеність; може бути використаний для розробки майже будь-якого продукту (web, desktop, etc.).

Недоліки: складний у написанні; високий поріг входження; низькорівнева; багатослівна у порівнянні з Python (через низькорівневість); невелика кількість бібліотек для ШІ, МН;

Java – об’єктно-орієнтована мова програмування, випущена компанією Sun Microsystems у 1995 році як основний компонент платформи Java [23].

Переваги: високорівнева; висока швидкодія; дуже хороша захищеність; може бути використаний для розробки майже будь-якого продукту (web, desktop, etc.).

Недоліки: складний у написанні; високий поріг входження; багатослівна у порівнянні з Python; невелика кількість бібліотек для ШІ, МН.

3.2 Програмна реалізація серверної частини програмного модуля автоматизованого тестування WEB-додатків

Серверна частина інформаційної технології складається з двох етапів:

1) реєстрація/авторизація користувача з боку клієнтської частини та переробка інформації на боці сервера. Таким чином це дає користувачеві доступ до перегляду/редагування власної інформації. Дану частину можливо відобразити наступною схемою (рис. 3.1).

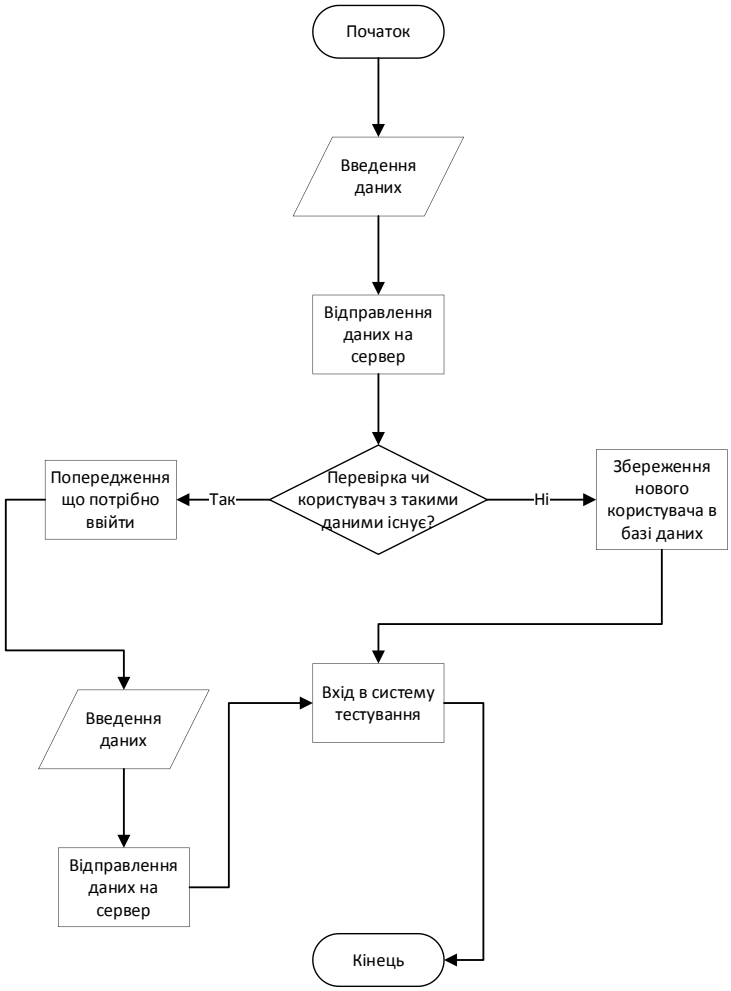


Рисунок 3.1 – Схема алгоритму реєстрації та входу користувача в систему тестувань

Нижче наведено функції які виконують дії такі як авторизація та реєстрація:

```
def register_user(post_data: dict):
    email = post_data.get('email')
    user = User.objects(email=email).first()
    if not user:
        logger.debug(f'Register a new user email={email}')

        user = User(
            email=email,
            password=post_data.get('password'),
            nickname=post_data.get('nickname')
        )
    else:
        response_object = {
            'message': 'User already exists. Please log in'
        }
        logger.error(response_object)
        return {'errors': response_object}
    try:
        user.save()
        return user.to_dict()
    except (PyMongoError, SMTPResponseException) as ex:
        user.clean()
        raise AppBackendError(ex).
```

Функція реєстрації приймає на вхід данні в вигляді словника з ключами почтова скринька, пароль та ім’я користувача. Після отримання даних йде перевірка на БД (база даних) чи існує користувач, якщо ні створюється клас User який є підкласом моделі БД за допомогою якого виконується збереження та відображення даних:

```
Def login_user(post_data: dict):
    user = User.objects(email=post_data.get(‘email’)).first()
    if not user:
        raise UnknownUserEmailError(email=post_data.get(‘email’))
    if user.verify_password(post_data.get(‘password’)):
        auth_token = user.encode_auth_token(user.id)
        if auth_token:
            response_object = {
                ‘auth_token’: auth_token,
                ‘duration’: 7200,
                ‘user’: {
                    ‘id’: str(user.id),
                    ‘is_admin’: user.is_admin,
                    ‘nickname’: user.nickname,
                    ‘email’: user.email
                }
            }
            client = Client(user.email, access_token=auth_token)
            oauth._client = client
            return response_object
    else:
```

```
response_object = {
    'message': 'Incorrect password.'
}
logger.error(response_object)
return {'errors': response_object}.
```

Функція авторизації на вході приймає словник зі почтовою скринькою та паролем, які потім перевіряються в бд та створюється ключ який діє 2 години, після кінчення сроку ключа потрібно повторна авторизація. На рисунку 3.2 наведена діаграма класу Users.

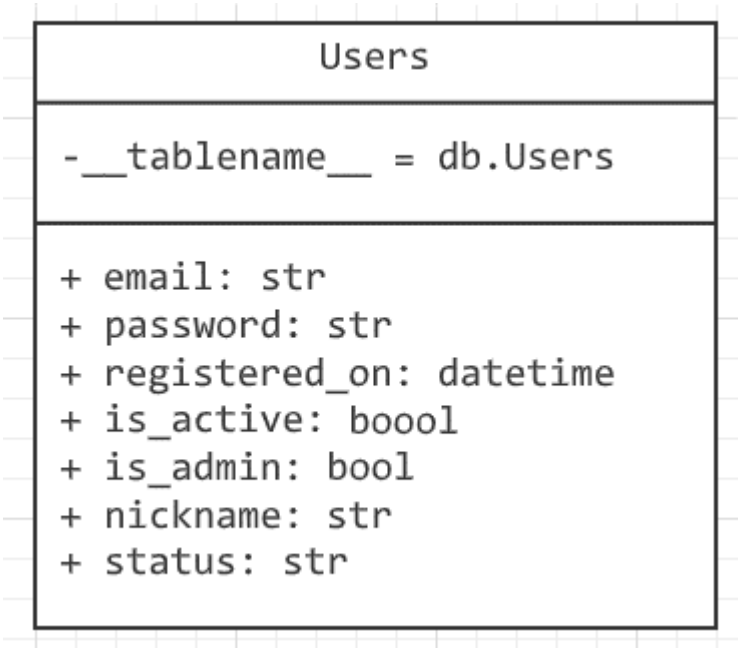


Рисунок 3.2 – Схема діаграми класу «Користувача»

2) додавання/змінення тестів до розкладу та запуск тестів:

Функція додавання тестів бере на вхід словник з такими ключами, як адреса сайту та період в продовж якого буде повторюватись його тестування. Перед тестуванням користувач створює запис в базі з вхідними даними та зберігає його.

```
@staticmethod
def add_new(data: dict):
    """
    Add new schedule

    :param dict data: Post Data from client
    :return: New Schedule
    :rtype: Schedules
```

```
        """
        User.register_delete_rule(User, 'schedule', 3)
        url = data.get('url')
        period = datetime.datetime.strptime(data.get('period'), '%Y-%m-%d %X')
        days, seconds, microseconds = Schedules.to_period(datetime.datetime.now(),
    period)
        obj = Schedules(
            url=url,
            user=g.user,
            started_at=datetime.datetime.now() + datetime.timedelta(days=days,
    seconds=seconds,

    microseconds=microseconds),
            days=days,
            seconds=seconds,
            microseconds=microseconds,
            report=[],
            repeat_count=0
        )
        obj.save()
        logger.warn(f'Added a new Schedule : {obj.__repr__()}')
        return obj.
```

Функція оновлення конкретного запису по первинному ключу. Перший параметр raw_id – це первинний ключ таблиці, raw – словник з ключами відповідними назві колонок таблиці. Після отримання вхідних даних ми шукаємо запис в таблиці та передаємо його класовим об’єктом, який потім оновлюємо в коді та зберігаємо вже з новими даними:

```
@classmethod
def upd_by_id(cls, rec_id, raw):
    """
    Update document by rec_id

    Args:
        rec_id (str): Document id
        raw (dict): Dict with keys to update

    Returns:
       Schedules : Schedule
    """
    obj = cls.get_by_id(rec_id)
    p = raw.get('period', None)
    period = datetime.datetime.strptime(p, '%Y-%m-%d %X') if p else
datetime.datetime.now() + datetime.timedelta(
        days=obj.days, seconds=obj.seconds,
        microseconds=obj.microseconds)
    days, seconds, microseconds = Schedules.to_period(datetime.datetime.now(),
period)
    for r in raw:
        new_raw = {r: raw[r]}
        if hasattr(obj, r):
            data = new_raw[r]
            if r in ('complete', 'is_active', 'repeat'):
```

```
                data = str_to_bool(new_raw[r])
            setattr(obj, r, data)
        if r == 'period':
            new_raw['days'] = days
            new_raw['seconds'] = seconds
            new_raw['microseconds'] = microseconds
            new_raw['started_at'] = period
            for ri in new_raw:
                setattr(obj, ri, new_raw[ri])
    obj.save()
    return obj.
```

Нижче на рисунку 3.2 наведена діаграма самого класу розкладу запуску тестів (Schedules) та його зв'язок із іншими класами такими як Користувач та Репорти:

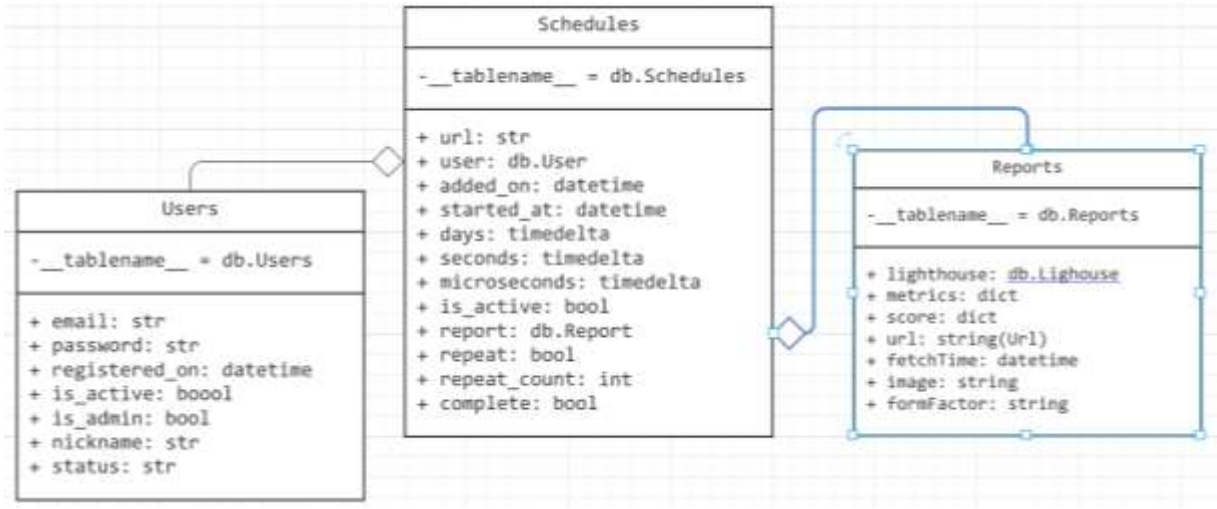


Рисунок 3.2 – Діаграма класів Schedules, Users, Reports

3.3 Тестування серверної частини програмного модуля автоматизованого тестування WEB-додатків

Тестування було проведено на програмному інтерфейсі Swagger. Протестовано основні модулі роботи програми, результати наведені на рисунках 3.3–3.8.

Після того як сервер завантажився, переходим за адресою http://127.0.0.1:8080 і показується сторінка прикладного програмного інтерфейсу (API).

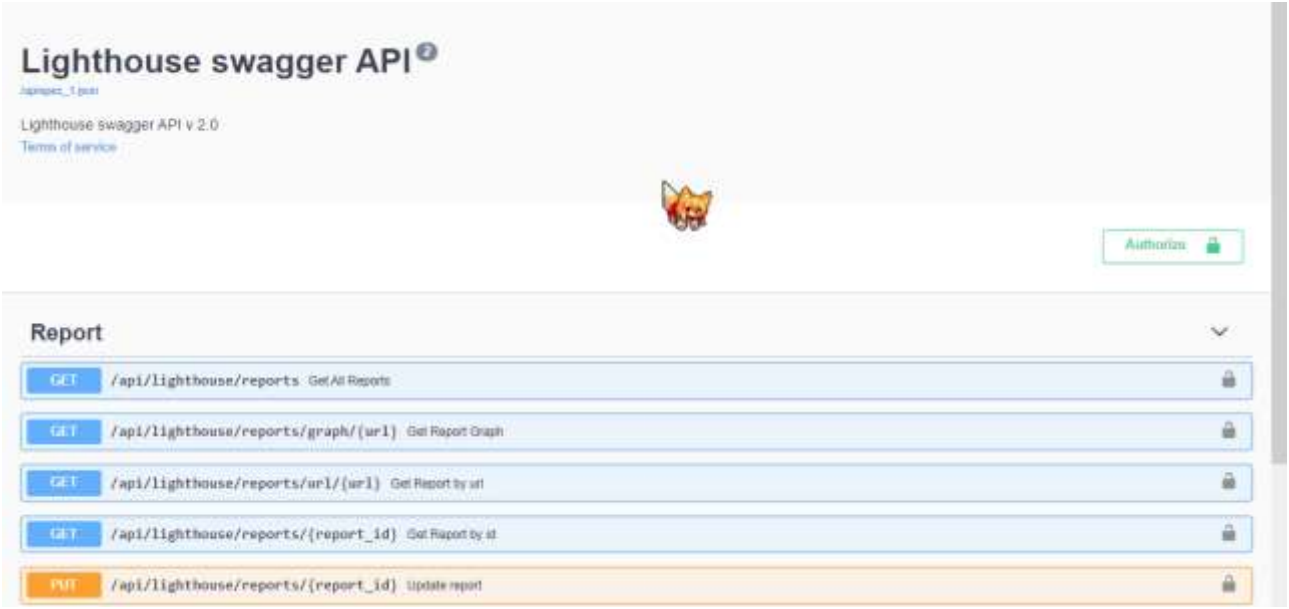


Рисунок 3.3 – Загальний вигляд прикладного програмного інтерфейс інформаційної технології

Далі провіримо тестування системи реєстрації та авторизації на API:

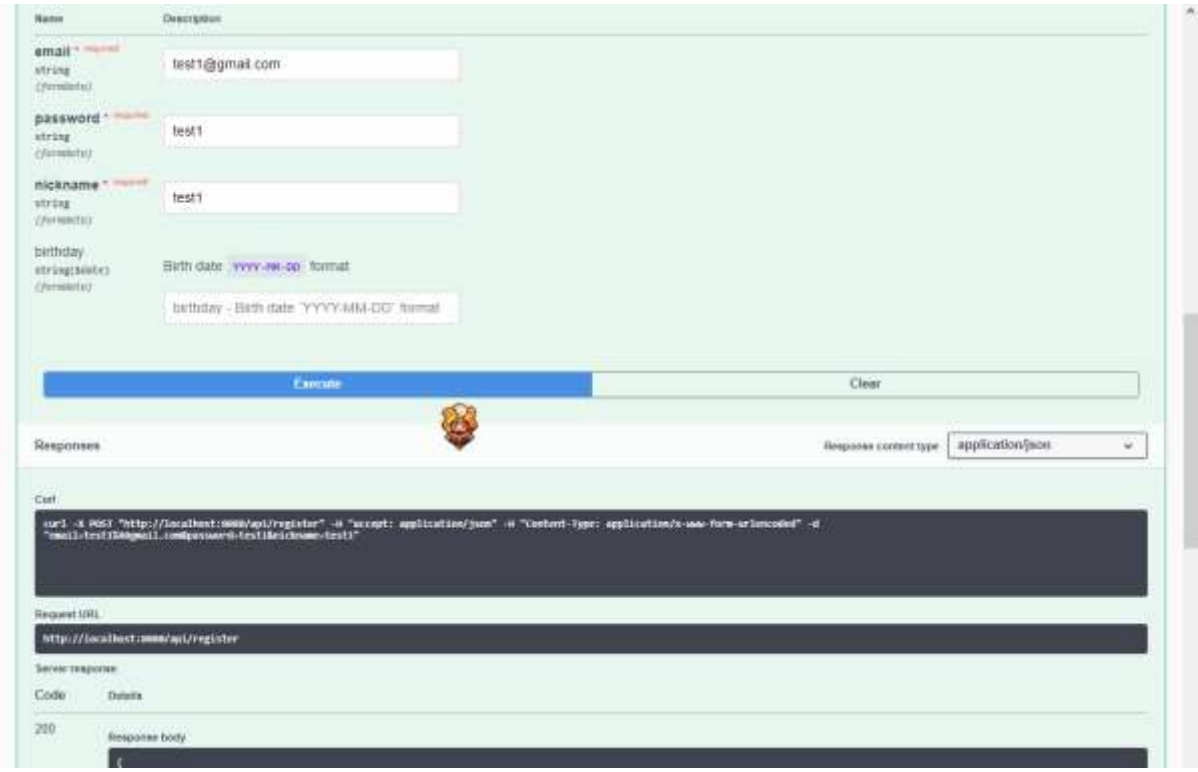


Рисунок 3.4 – Загальний вигляд інтерфейсу ідентифікацій користувача за допомогою API (/api/register)

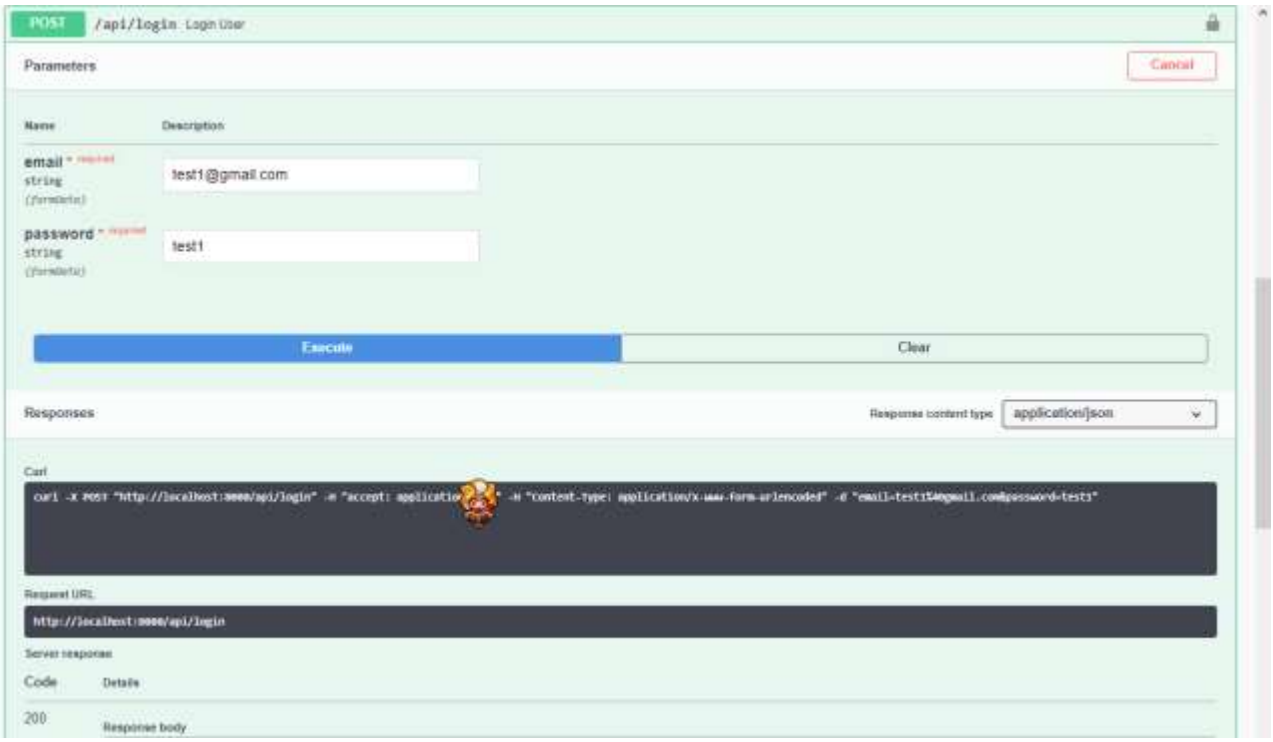


Рисунок 3.5 – Загальний вигляд інтерфейсу ідентифікацій користувача за допомогою API (/api/login)

Якщо користувач не зареєстрований в системі, переходимо в розділ User Authentication.

Для реєстрації користувача потрібно заповнити поля «Email, Password, Nickname», після чого натискаєм кнопку “Execute” який відправляє запит і якщо користувач з такою поштою не зареєстрований тоді реєстрація проходить успішно, після чого потрібно авторизуватись.

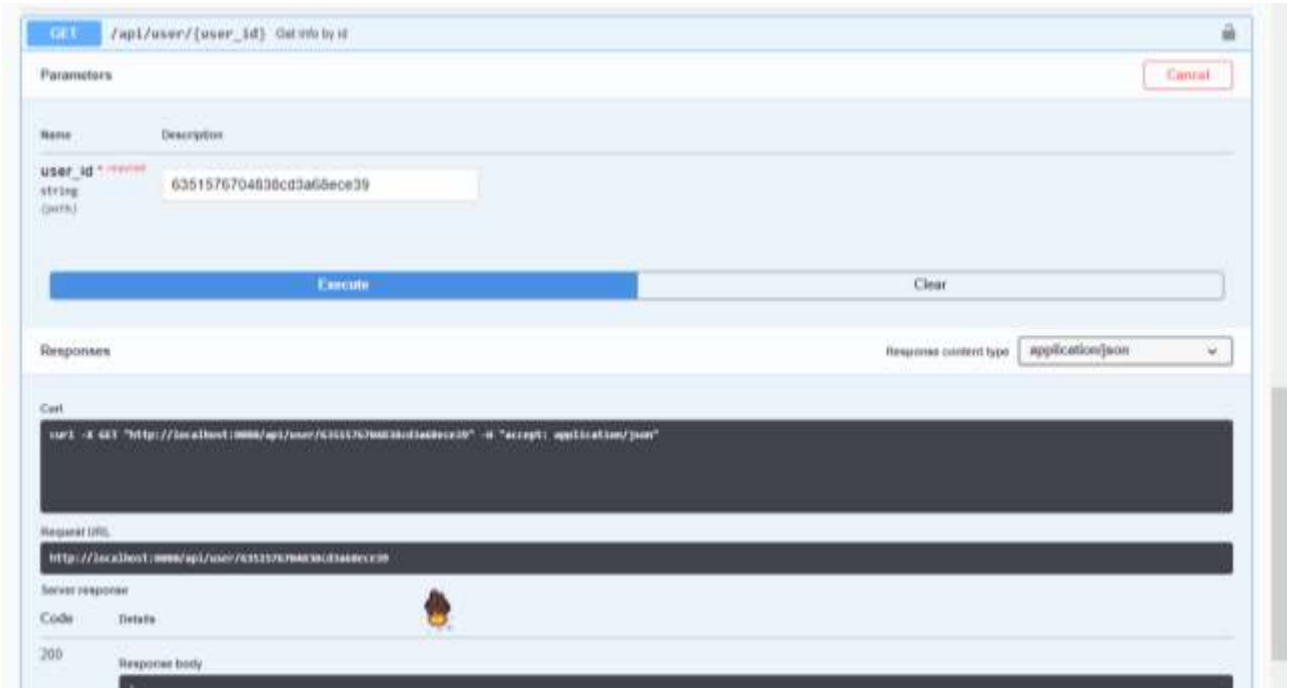


Рисунок 3.6 – Загальний вигляд інтерфейсу даних користувача за допомогою API (/api/user/<user_id>)

Протестуємо захист від неавторизованих користувачів. Він не дає змогу редагувати, створювати дані в базі даних. Як видно нижче на рисунку 3.11 при відправці даних неавторизованим користувачем спливає помилковий запит 401

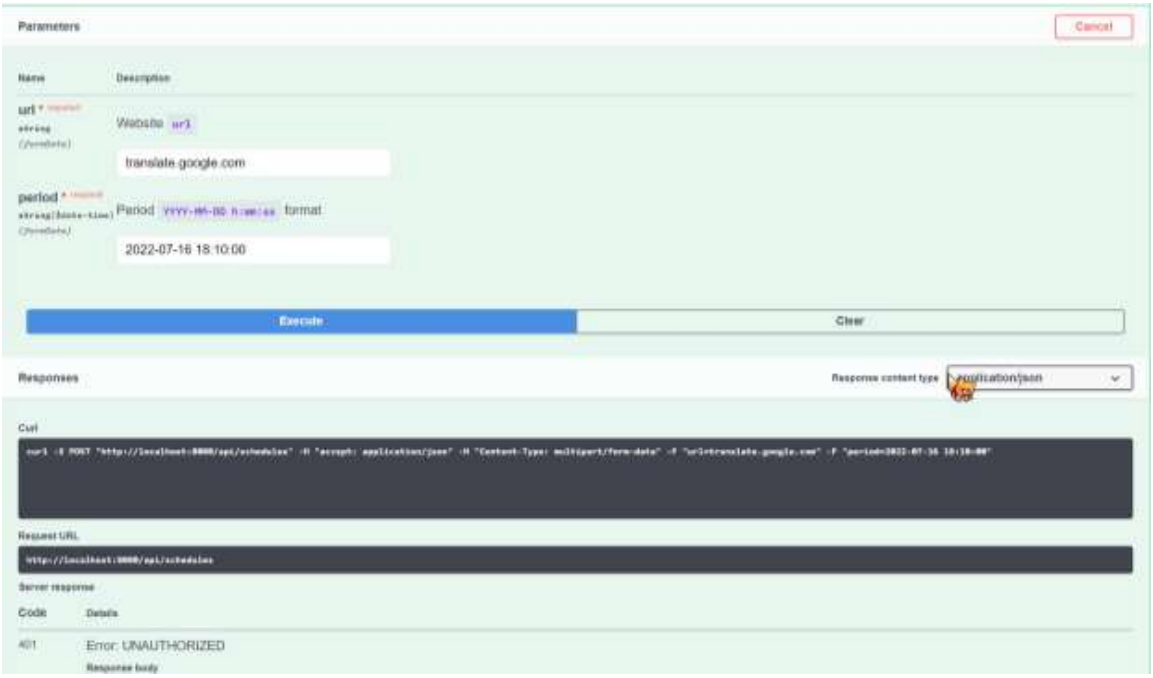


Рисунок 3.7 – Загальний вигляд захисту від неавторизованих користувачів

У меті була поставлена ціль підвищити рівень відповідності вимогам забезпечення якості програмного забезпечення за допомогою розробки інформаційної технології автоматизованого тестування WEB-додатків на серверному рівні. Для перевірки її досягнення потрібно порівняти характеристики розробленої інформаційної технології автоматизованого тестування WEB-додатків з існуючими аналогами. Показники надійності серверної частини зображені на рисунку 3.8.

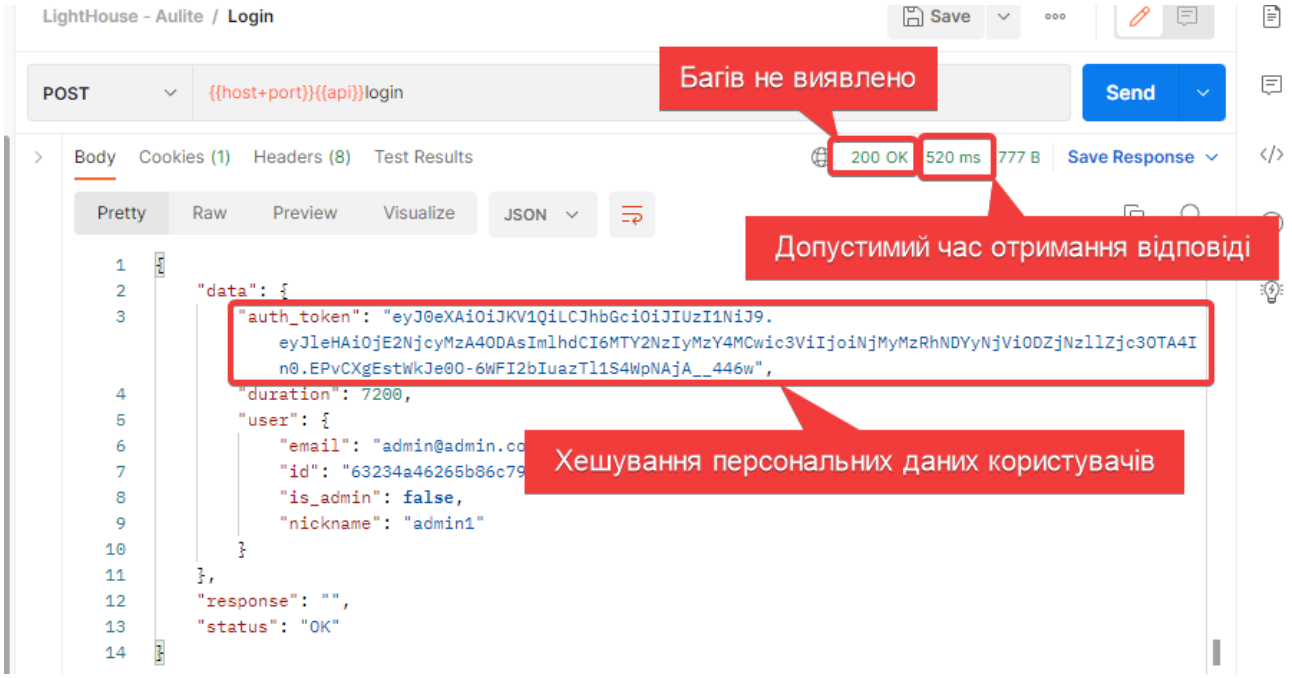


Рисунок 3.8 – Загальний вигляд інтерфейсного вікна із показниками надійності серверної системи

Ймовірність виникнення збою роботи станове менше 50%; допустимий час отримання відповіді від сервера – 520 мс; степінь хешування персональних даних складає більше 32 одиниці.

3.4 Висновок до розділу 3

У даному розділі обґрунтовано вибір інструментарію, що використовувався при розробці клієнтської частини системи автоматизованого тестування WEB-додатків.

Описана загальна структурна схема функціонування клієнтської частини системи автоматизованого тестування WEB-додатків. Наведений та описаний алгоритм роботи розробленого методу автоматизованого тестування WEB-додатків.

Спроектований схематичний вигляд інтерфейсу головної сторінки системи автоматизованого тестування WEB-додатків.

Було проведено тестування розробленої клієнтської частини системи автоматизованого тестування WEB-додатків.

Порівнюючи із розглянутими у першому розділі аналогами, середнє значення характеристик надійності розробленої інформаційної технології автоматизованого тестування WEB-додатків, є вищим, а саме у 1,32 рази надійніше за JMeter, у 1,34 рази надійніше за TestCafe та у 3,23 рази надійніше за ODT. Тому доцільно вважати що мета досягнута.

4 ЕКОНОМІЧНА ЧАСТИНА

4.1 Комерційний та технологічний аудит науково-технічної розробки

Метою даного розділу магістерської кваліфікаційної роботи є проведення технологічного аудиту, в даному випадку нового програмного продукту розробки автоматизованого тестування WEB-додатків.

Аналогом може бути програмна система Selenium, основним недоліком якої є відсутність надійності персональних даних. Щодо вартості, то це апаратно-програмний комплекс коштує десятки тис. доларів.

Для проведення комерційного та технологічного аудиту залучають не менше 3-х незалежних експертів. Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням п’ятибальної системи оцінювання за 12-ма критеріями, у відповідності із таблиці 4.1.

Таблиця 4.1 – Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Бали (за 5-ти бальною шкалою)					
Кри-терій	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги					
2	Багато аналогів на малому ринку	Ринкові п Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку

Продовження табл. 4.1

3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практик на здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так із комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно ви-

Продовження табл. 4.1

11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Усі дані по кожному параметру занесено в таблиці 4.2

Таблиця 4.2 – Результати оцінювання комерційного потенціалу розробки

Критерії оцінювання	ПІБ експертів		
	Експерт 1	Експерт 2	Експерт 3
	Бали		
Технічна здійсненність концепції	4	3	4
Наявність аналогів на ринку	4	4	4
Цінова політика	3	4	4
Технічні та споживчі властивості виробу	4	3	4
Експлуатаційні витрати	3	4	4
Ринок збуту	4	3	3
Конкурентоспроможність	3	4	4
Фахівці з технічної і комерційної реалізації	4	3	4
Фінансування	4	4	3
Матеріально-технічна база	4	4	3
Термін реалізації ідеї	4	4	3
Супровідна документація	4	3	4
Сума	45	43	44
Середньоарифметична сума балів	(45+43+44) / 3 = 44		

За даними таблиці 4.2 можна зробити висновок щодо рівня комерційного потенціалу даної розробки. Для цього доцільно скористатись рекомендаціями, наведеними в таблиці 4.3.

Таблиця 4.3 – Рівні комерційного потенціалу розробки

Середньоарифметична сума балів СБ, розрахована на основі	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

Провівши дослідження рівня комерційного потенціалу розробки «Інформаційна технологія автоматизованого тестування WEB-додатків. Серверна частина» виявлено, що його рівень високий та становить 44 бали. Перевагами даної розробки є можливість автоматизації тестування WEB-додатків та забезпечення стабільності роботи WEB-додатку.

Проведемо оцінювання рівня конкурентоспроможності розробки.

Конкурентоспроможність розкривається через систему якісних та економічних показників.

Якісні показники конкурентоспроможності характеризують властивості розробки, завдяки яким вона задовольняє конкретні потреби (нормативні та технічні).

Економічні показники конкурентоспроможності характеризують сумарні витрати споживачів на задоволення їх потреб цією розробкою. Вони складаються з витрат на придбання (ціна продажу) і витрат, пов’язаних з експлуатацією виробу.

Оцінювання рівня конкурентоспроможності науково-технічної розробки здійснюється в декілька етапів.

Етап 1. Розрахунок одиничних параметричних індексів

Процедура визначення якісних одиничних параметричних індексів за технічними показниками (показниками якості) здійснюється за відповідними формулами.

Якщо зменшення величини параметра свідчить про підвищення якості нової розробки, то одиничний параметричний індекс розраховується за оберненою формулою:

$$q_i = \frac{P_i}{P_{\text{базі}}}.$$

де $q_{\text{і}}$ – одиничний параметричний індекс, розрахований за і-м параметром; $P_{\text{і}}$ – значення і-го параметра розробки; $P_{\text{базі}}$ – аналогічний параметр базової розробки-аналога, з якою проводиться порівняння.

Відповідно використавши дані параметрів аналога з розділу 1, та дані параметрів інформаційної технології автоматизованого тестування WEB-додатків з розділу 3. Для прикладу обрахуємо значення першого параметру, а усі дані по кожному параметру занесемо в таблицю 4.4.

$$q_i = \frac{5}{15} = 0{,}33$$

Таблиця 4.4 – Значення параметрів програми-аналога «Selenium» та інформаційної технології автоматизованого тестування WEB-додатків.

Параметр	Аналог	Нова розробка	Індекс зміни значення параметра	Коефіцієнт вагомості
Технічні				
Швидкість транзакцій	5	15	0,33	0,25
Кількость запитів від користувача	10	5	2,00	0,11
Швидкість обробки даних	20	50	0,40	0,19
Швидкість обміну даними	100	125	0,80	0,24
Допустимий час отриманя відповіді	10	5	2,00	0,6
Степінь хешування персональних даних	16	32	0,50	0,2
Економічні				
Річні експлуатаційні витрати на підтримку системи тестування (грн)	950	500	0,53	0,4
Вартість річної підписки (для клієнта) (грн)	412	275	0,67	0,6

Нормативні параметри оцінюються показником, який отримує одне з двох значень: 1 – товар відповідає нормам і стандартам; 0 – не відповідає.

Груповий показник конкурентоспроможності за нормативними параметрами розраховується як добуток частинних показників за кожним параметром за формулою:

$$\mathrm{I}_{\mathrm{НП}} = \prod_{i=1}^{n} q_i,$$

де $I_{\mathrm{НП}}$ – загальний показник конкурентоспроможності за нормативними параметрами; q_{i} – одиничний (частинний) показник за і-м нормативним параметром; n – кількість нормативних параметрів, які підлягають оцінюванню.

За нормативними параметрами інформаційна технологія, що розробляється, відповідає вимогам ДСТУ, тому $I_{\mathrm{НП}} = 1$.

Значення групового параметричного індексу за технічними параметрами визначається з урахуванням вагомості (частки) кожного параметра:

$$\mathrm{I}_{\mathrm{ТП}} = \sum_{i=1}^{n} q_i \cdot \alpha_i,$$

де $I_{\mathrm{тп}}$ – груповий параметричний індекс за технічними показниками (порівняно з аналогом); q_{i} – одиничний параметричний показник і-го параметра; α_{i} – вагомість і-го параметричного показника, $\sum_{i=1}^{n} \alpha_i = 1$; n – кількість технічних параметрів, за якими оцінюється конкурентоспроможність.

$$\mathrm{I}_{\mathrm{ТП}} = 0.33 * 0.25 + 2 * 0.11 + 0.40 * 0.19 + 0.80 * 0.24 + 2 * 0{,}6 + 0.5 * 0.2 = 1.87$$

Груповий параметричний індекс за економічними параметрами розраховуємо за формулою:

$$\mathrm{I}_{\mathrm{ЕП}} = \sum_{i=1}^{m} q_i \cdot \beta_i,$$

де $I_{\mathrm{ЕП}}$ – груповий параметричний індекс за економічними показниками; q_{i} – економічний параметр і-го виду; β_{i} – частка і-го економічного параметра,

$\sum_{i=1}^{m} \beta_i = 1$; m – кількість економічних параметрів, за якими здійснюється оцінювання.

Проведемо аналіз параметрів згідно даних таблиці.

$$\text{I}_{\text{ЕП}} = 0{,}53 * 0{,}4 + 0{,}67 * 0{,}6 = 0{,}61.$$

Етап 3. Розрахунок інтегрального показника.

На основі групових параметричних індексів за нормативними, технічними та економічними показниками розраховують інтегральний показник конкурентоспроможності за формулою:

$$\text{К}_{\text{ІНТ}} = \text{I}_{\text{НП}} \cdot \frac{\text{I}_{\text{ТП}}}{\text{I}_{\text{ЕП}}}$$

$$\text{К}_{\text{ІНТ}} = 1 * \frac{1{,}87}{0{,}61} = 3{,}06.$$

Інтегральний показник конкурентоспроможності $К_{\text{ІНТ}} > 1$, отже розробка переважає відомі аналоги за своїми техніко-економічними показниками.

4.2 Прогнозування витрат на виконання науково-дослідної роботи

Основна заробітна плата розробників, яка розраховується за формулою:

$$З_o = \sum_i \frac{k \times M_{ni} \times t_i}{T_p}, \quad (4.1)$$

$$З_o = \sum_i \frac{k \times M_{ni} \times t_i}{T_p} = \frac{1 \times 25000 \times 33}{22} + \frac{1 \times 21000 \times 33}{22} = 37\,500{,}00 +$$

$$31\,500{,}00 = \ 69\,000{,}00 \text{ (грн.)}$$

де k – кількість посад розробників, залучених до процесу розробки; $М_{пі}$ – місячний посадовий оклад конкретного розробника, грн; t_i – кількість днів роботи конкретного розробника, дн.; $Т_р$ – середня кількість робочих днів в місяці, $Т_р$ =21…23 дні.

Результати розрахунків зведемо до таблиці 4.4.

Таблиця 4.4 – Основна заробітна плата розробників

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник проекту	25000	1136,36	33	60 606,06
Інженер	21000	954,55	33	50 909,09
Всього				111 515, 15

Так як в даному випадку розробляється програмний продукт, то розробник виступає одночасно і основним робітником, і тестувальником розроблюваного програмного продукту.

Додаткова заробітна плату розробників, які приймали участь в розробці інформаційної технології, прийнято розраховувати як 10…12% від основної заробітної плати розробників та робітників:

$$З_д = З_о \cdot \frac{11\%}{100\%}, \quad (4.2)$$

$$З_д = \left(111\,515{,}15 \cdot \frac{11\%}{100\%}\right) = 12\,266{,}00 \text{ (грн.)}.$$

Нарахування на заробітну плату розробників. Згідно діючого законодавства нарахування на заробітну плату складають 22 % від суми основної та додаткової заробітної плати:

$$H_з = \left(З_о + З_р + З_д\right) \cdot \frac{22\%}{100\%}, \quad (4.3)$$

$$H_з = (115\,515{,}15 + 12\,266{,}67) \cdot \frac{22\%}{100\%} = 27\,232{,}00 \text{ (грн.)}.$$

Оскільки для розроблювального пристрою не потрібно витрачати матеріали та комплектуючи, то витрати на матеріали і комплектуючі дорівнюють нулю.

Амортизація обладнання, що використовувалось для розробки в спрощеному вигляді амортизація обладнання, що використовувалась для розробки розраховується за формулою:

$$A = \frac{Ц}{T_в} \cdot \frac{t_{вик}}{12} \text{ (грн.)}. \quad (4.4)$$

де *Ц* – балансова вартість обладнання, грн.; *Т* – термін корисного використання обладнання згідно податкового законодавства, років; $t_{вик}$ – термін використання під час розробки, місяців

Розрахуємо, для прикладу, амортизаційні витрати на ноутбук балансова вартість якого становить 32 000 грн., термін його корисного використання згідно податкового законодавства – 2 роки, а термін його фактичного використання – 2 місяця:

$$А_{обл} = \frac{32\,000}{2} \times \frac{2}{12} = 2\,666{,}67 \text{ грн.}$$

Аналогічно визначаємо амортизаційні витрати на інше обладнання та приміщення. Розрахунки заносимо до таблиці 4.5. Для розрахунку амортизації нематеріальних ресурсів використовується формула:

$$А_{н.р.} = Ц_{Н.р.} * Н_а * \frac{t_{вик}}{12}. \quad (4.5)$$

Розрахуємо, для прикладу, амортизаційні витрати на ліцензію оперативної системи балансова вартість якого становить 13 500 грн., термін його корисного використання згідно податкового законодавства – 1,55 роки:

$$А_{н.р.} = Ц_{Н.р.} * Н_а * \frac{t_{вик}}{12} = 13\,500 * 10\% * \frac{2}{12} = 225$$

Норма амортизації H_a приймемо за 10 %.

Таблиця 4.5 – Амортизаційні відрахування матеріальних і нематеріальних ресурсів для розробників

Найменування обладнання	Балансова вартість, грн.	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн.
Ноутбук (Acer Nitro 5)	32 000,00	2,00	2,00	2 666,67
Приміщення	500 000,00	20,00	2,00	4 166,67
Сервер	45 000,00	2,00	2,00	3 750,00
Ліцензійна ОС, та спеціалізовані ліцензійні нематеріальні ресурси	13 500,00	-	2,00	225,00
Всього				11 033,33

Тарифи на електроенергію для непобутових споживачів (промислових підприємств) відмінні від тарифів на електроенергію для населення. При цьому тарифи на розподіл електроенергії у різних постачальників (енергорозподільних компаній), будуть різними. Крім того, розмір тарифу

залежить від класу напруги (1-й або 2-й клас). Тарифи на розподіл електроенергії для всіх енергорозподільних компаній встановлює Національна комісія з регулювання енергетики і комунальних послуг (НКРЕКП). Витрати на силову електроенергію розраховуються за формулою:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \times t_i \times Ц_e \times К_{впі}}{\eta_i}, \quad (4.6)$$

де W_{yi} – встановлена потужність обладнання на певному етапі розробки, кВт; t_i – тривалість роботи обладнання на етапі дослідження, год; $Ц_e$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії); $К_{впі}$ – коефіцієнт, що враховує використання потужності, $К_{впі}$ <1; η_i – коефіцієнт корисної дії обладнання, η_i<1.

$$Ц_e = \left(Ц_{опт} + Ц_{розп} + Ц_{пост}\right)\left(1 + \frac{ПДВ}{100\%}\right), \quad (4.6)$$

$$Ц_e = (3.411 + 1.257 + 0.346) * 1.2 = 6.02 \text{ грн.}$$

де $Ц_{опт}$ - середня оптова ціна електроенергії, яка визначається оператором ринку (без ПДВ), грн за 1 кВт·год; $Ц_{розп}$ - вартість розподілу електроенергії окремою енергорозподільчою компанією (без *ПДВ*), грн за 1 кВт·год; $Ц_{пост}$ - вартість постачання електроенергії від енергорозподільчої компанії до конкретного споживача (без *ПДВ*), грн за 1 кВт·год; *ПДВ* - величина податку на додану вартість, %, у 2022 році *ПДВ*=20%.

Розрахуємо, для прикладу, Витрати на електроенергію для комп'ютера потужність якого становить 0,09 кВт, тривалість роботи за 33 повних робочих днів – 264 години, а коефіцієнт, що враховує використання потужності – 0,8 кВт/год..

$$B_{ек} = W_{yi} \times t_i \times Ц_е \times К_{впі} = 0{,}09 * 264 * 6{,}02 * 0{,}8 = 114{,}43 \text{ грн.}$$

Аналогічно визначаємо витрати на електроенергію на інше обладнання та приміщення. Розрахунки заносимо до таблиці 4.6.

Таблиця 4.6 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Ноутбук (Acer Nitro 5)	0,09	33*8=264,00	114,43
Сервер	0,12	33*24=792,00	474,88
Освітлення	0,029	33*8=264,00	43,78
Всього			633,09

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками. Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників:

$$І_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (4.7)$$

де $H_{ів}$ – норма нарахування за статтею «Інші витрати».

$$І_в = (115\,515{,}15 + 0) * \frac{115\%}{100\%} = 128\,242{,}42 \text{ (грн.)}.$$

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-

технічну інформацію та рекламу та ін. Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 100…150% від суми основної заробітної плати дослідників:

$$Н_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (4.8)$$

де $H_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати».

$$Н_{нзв} = 115\,515.15 * \frac{126\%}{100\%} = 140\,509.09 \text{ (грн.)}.$$

Сума всіх попередніх статей витрат дає загальні витрати на проведення науково-дослідної роботи:

$$В_{заг} = З_о + З_{дод} + H_з + A_{обл} + В_е + I_в + H_{нзв}. \quad (4.9)$$

$$В_{заг} = 115\,515{,}15 + 12\,266{,}67 + 27\,232{,}00 + 11\,033{,}00 + 633{,}09 + 128\,242{,}42 + 140\,509.09 = 420\,398{,}42 \text{ грн.}$$

Загальні витрати на завершення науково-технічної роботи та оформлення її результатів розраховуються *ЗВ*, визначається за формулою:

$$ЗВ = \frac{В_{заг}}{\eta} \text{ (грн)}, \quad (4.10)$$

де η – коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи.

Так, якщо науково-технічна розробка знаходиться на стадії: науково-дослідних робіт, то η=0,1; технічного проектування, то η=0,2; розробки

конструкторської документації, то η=0,3; розробки технологій, то η=0,4; розробки дослідного зразка, то η=0,5; розробки промислового зразка, то η=0,7; впровадження, то η=0,9. Оберемо η = 0,5, так як розробка, на даний момент, знаходиться на стадії розробки дослідного зразка:

$$ЗВ = \frac{420\,398.42}{0,5} = 840\,796,85 \text{ грн.}$$

4.3 Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором

У ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку. Саме зростання чистого прибутку забезпечить потенційному інвестору надходження додаткових коштів, дозволить покращити фінансові результати його діяльності, підвищить конкурентоспроможність та може позитивно вплинути на ухвалення рішення щодо комерціалізації цієї розробки.

Для того, щоб розрахувати можливе зростання чистого прибутку у потенційного інвестора від можливого впровадження науково-технічної розробки необхідно:

а) вказати, з якого часу можуть бути впроваджені результати науково-технічної розробки;

б) зазначити, протягом скількох років після впровадження цієї науково-технічної розробки очікуються основні позитивні результати для потенційного інвестора (наприклад, протягом 3-х років після її впровадження);

в) кількісно оцінити величину існуючого та майбутнього попиту на цю або аналогічні чи подібні науково-технічні розробки та назвати основних суб’єктів (зацікавлених осіб) цього попиту;

г) визначити ціну реалізації на ринку науково-технічних розробок з аналогічними чи подібними функціями.

При розрахунку економічної ефективності потрібно обов'язково враховувати зміну вартості грошей у часі, оскільки від вкладення інвестицій до отримання прибутку минає чимало часу. При оцінюванні ефективності інноваційних проектів передбачається розрахунок таких важливих показників:

- абсолютного економічного ефекту (чистого дисконтованого доходу);
- внутрішньої економічної дохідності (внутрішньої норми дохідності);
- терміну окупності (дисконтованого терміну окупності).

Аналізуючи напрямки проведення науково-технічних розробок, розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором можна об'єднати, враховуючи визначені ситуації з відповідними умовами.

Розробка чи суттєве вдосконалення програмного продукту для використання масовим споживачем.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

$$\Delta П_i = (\pm \Delta Ц_0 * N + Ц_0 * \Delta N)_i * \lambda * \rho * \left(1 - \frac{\vartheta}{100}\right), \qquad (4.11)$$

де $\pm\Delta Ц_0$ – зміна вартості програмного продукту (зростання чи зниження) від впровадження результатів науково-технічної розробки в аналізовані періоди часу; N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки; $Ц_о$ – основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки, $Ц_о = Ц_б \pm \Delta Ц_о$; $Ц_б$ – вартість програмного продукту у році до впровадження результатів розробки; ΔN – збільшення кількості споживачів продукту, в аналізовані періоди часу, від покращення його певних характеристик; λ – коефіцієнт, який враховує сплату

податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт λ = 0,8333; p – коефіцієнт, який враховує рентабельність продукту; ϑ – ставка податку на прибуток, у 2022 році ϑ =18%.

Припустимо, що при прогнозованій ціні 1 000 грн. за одиницю виробу, термін збільшення прибутку складе 3 роки. Після завершення розробки і її вдосконалення, можна буде підняти його ціну на 500 грн. Кількість одиниць реалізованої продукції також збільшиться: протягом першого року – на 10 000 шт., протягом другого року – на 15 000 шт., протягом третього року на 20 000 шт. До моменту впровадження результатів наукової розробки реалізації продукту не було:

$$\Delta\text{П}_1 = (500 * 0 + (1\,000 + 500) * 10\,000) * 0{,}8333 * 0{,}45 * (1 - 0{,}18) = 4\,612\,315{,}50 \text{ грн.}$$

$$\Delta\text{П}_2 = \big(500 * 0 + (1\,000 + 500) * (10\,000 + 15\,000)\big) * 0{,}8333 * 0{,}45 * (1 - 0{,}18) = 11\,530\,788{,}75 \text{ грн.}$$

$$\Delta\text{П}_3 = \big(500 * 0 + (1\,000 + 500) * (10\,000 + 15\,000 + 20\,000)\big) * 0{,}8333 * 0{,}45 * (1 - 0{,}18) = 20\,755\,419{,}75 \text{ грн.}$$

Отже, комерційний ефект від реалізації результатів розробки за три роки складе 36 898 524,00 грн.

Розраховуємо приведену вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$\text{ПП} = \sum_{1}^{\text{T}} \frac{\Delta\text{П}_\text{i}}{(1+\tau)^t}, \tag{4.12}$$

де $\Delta П_i$ −збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої науково-дослідної (науково-технічної) роботи, грн; T − період часу, протягом якою виявляються результати впровадженої науково-дослідної (науково-технічної) роботи, роки; τ −ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ = 0,05...0,15; t – період часу (в роках).

Збільшення прибутку ми отримаємо починаючи з першого року:

$$\text{ПП} = \left(\frac{4\,612\,315{,}50}{(1+0{,}1)^1}\right) + \left(\frac{11\,530\,788{,}75}{(1+0{,}1)^2}\right) + \left(\frac{20\,755\,419{,}75}{(1+0{,}1)^3}\right)$$
$$= 4\,193\,014{,}09 + 9\,529\,577{,}48 + 15\,593\,854{,}06$$
$$= 29\,316\,445{,}63 \text{ грн.}$$

Далі розраховують величину початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки. Для цього можна використати формулу:

$$PV = k_{\text{інв}} * \text{ЗВ}, \qquad (4.13)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай $k_{інв}$=2…5, але може бути і більшим; $ЗВ$ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

$$PV = 2 * 1\,262\,365{,}53 = 2\,524\,731{,}05 \text{ грн.}$$

Тоді абсолютний економічний ефект $E_{\text{абс}}$ або чистий приведений дохід $(NPV, Net\ Present\ Value)$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$\text{Е}_{\text{абс}} = \text{ПП} - PV, \qquad (4.14)$$

$$E_{\text{абс}} = 29\,316\,445{,}63 - 2\,524\,731{,}05 = 26\,791\,714{,}58 \text{ грн.}$$

Оскільки $E_{абс} > 0$, то вкладання коштів на виконання та впровадження результатів даної науково-дослідної (науково-технічної) роботи може бути доцільним.

Для остаточного прийняття рішення з цього питання необхідно розрахувати внутрішню економічну дохідність або показник внутрішньої норми дохідності (IRR, Internal Rate of Return) вкладених інвестицій та порівняти її з так званою бар'єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку науково-технічну розробку вкладати буде економічно недоцільно.

Розрахуємо внутрішню економічну дохідність інвестицій E_B. Для цього використаємо формулу:

$$E_{\text{в}} = \sqrt[T_{\text{ж}}]{1 + \frac{E_{\text{абс}}}{PV}} - 1, \tag{4.15}$$

де $E_{\text{абс}}$ – абсолютний економічний ефект вкладених інвестицій, грн; PV – теперішня вартість початкових інвестицій, грн; $T_{\text{ж}}$ – життєвий цикл науково-технічної розробки, тобто час від початку

$$E_{\text{в}} = \sqrt[3]{1 + \frac{26\,791\,714{,}58}{2\,524\,731{,}05}} - 1 = 1{,}264$$

Визначимо мінімальну ставку дисконтування, за формулою:

$$\tau_{\text{мін}} = d + f, \tag{4.16}$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2022 році в Україні d = (0,9…0,12); f – показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,5).

$$\tau_{\text{мін}} = 0{,}5 + 0{,}3 = 0{,}8$$

Так як $E_в > \tau_{min}$, то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Далі розраховуємо період окупності інвестицій $T_{ок}$ (DPP, Discounted Payback Period), які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$\text{Т}_{\text{ок}} = \frac{1}{\text{Е}_{\text{в}}}, \qquad (4.17)$$

де $E_{\text{в}}$ – внутрішня економічна дохідність вкладених інвестицій.

$$\text{Т}_{\text{ок}} = \frac{1}{1.264} = 0{,}791 \text{ р.}$$

Оскільки $T_{ок}$ < 3-х років, а саме термін окупності рівний 0,791 роки, то фінансування даної наукової розробки є доцільним.

4.4 Висновок до розділу 4

Економічна частина даної роботи містить розрахунок витрат на розробку нового програмного продукту, сума яких складає – 840 796,85 гривень. Було спрогнозовано орієнтовану величину витрат по кожній з статей витрат. Також розраховано чистий прибуток, який може отримати виробник від реалізації нового технічного рішення, розраховано період окупності витрат для інвестора та економічний ефект при використанні даної розробки. В результаті аналізу розрахунків можна зробити висновок, що розроблений програмний продукт за ціною дешевший за аналог і є високо конкурентоспроможним. Період окупності складе близько 0,791 роки.

ВИСНОВКИ

У результати виконання магістерської кваліфікаційної роботи було проведено аналіз сучасного стану розвитку систем автоматизованого тестування WEB-додатків, на базі аналізу предметної області автоматизованого тестування WEB-додатків, аналізу відомих технічних рішень серверних модулів програмних систем автоматизованого тестування WEB-додатків та порівняльного аналізу характеристик серверних модулів програмних систем автоматизованого тестування WEB-додатків.

Було здійснено проектування інтелектуальної моделі системи автоматизованого тестування WEB-додатків, розроблено математичну модель «чорної скриньки» автоматизованого динамічного тестування WEB-додатків та метод «чорної скриньки» автоматизованого динамічного тестування WEB-додатків, проведено проектування інтелектуальної моделі системи автоматизованого тестування WEB-додатків.

Середнє значення характеристик надійності розробленої інформаційної технології автоматизованого тестування WEB-додатків, є вищим, а саме у 1.32 рази надійніше за JMeter, у 1.34 рази надійніше за TestCafe та у 3.23 рази надійніше за ODT.

Виконано розрахунок витрат на розробку нового програмного продукту, сума яких складає 503475,00 гривень. Було прогнозовано орієнтовану величину витрат по кожній з статей витрат. Також розраховано чистий прибуток, який може отримати виробник від реалізації нового технічного рішення, розраховано період окупності витрат для інвестора та економічний ефект при використанні даної розробки. У результаті аналізу розрахунків можна зробити висновок, що розроблений програмний продукт дешевший за аналог і є високо конкурентоспроможним. Період окупності складе близько 0,80 роки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Я. В. Іванчук, О. К. Галка Інформаційна технологія автоматичного тестування веб-додатків. Серверна частина / в Матеріали конференції «LI Науково-технічна конференція підрозділів Вінницького національного технічного університету (2022)», Вінниця, 2022. [Електронний ресурс]. Режим доступу:https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa2022/paper /view/15502.

2. Куликов Святослав Тестирование программного обеспечения. Базовый курс/ Святослав Куликов., 2021. – Т. 3 : – 298 с.

3. Луиза Тамре. Введение в тестирование программного обеспечения. — Вильямс, 2003. — 368 с.

4. Fewster M. Software Test Automation / M. Fewster, D. Graham. – ACM Press, 1999. – 600 p.

5. Web site about Tests Automation [Електроний ресурс] Режим доступу - http://automated-testing.info/tools

6. Samanta, S. K., Achilleos, A., Moiron, S. R. F., Woods,J., and Ghanbari, M. (2010) Automatic languagetranslation for mobile SMS, International Journal of Information Communication Technologies and HumanDevelopment (IJICTHD) 2, 43-58.12.

7. Sikuli: using GUI screenshots for search and automation / [Tom Yeh, Tsung-Hsiang Chang, Robert C. Miller] // In Proceedings of the 22nd annual ACM symposium on User interface software and technology (UIST '09). – ACM, NY, USA. – P. 183-192.

8. D. S. Rosenblum and E. J. Weyuker. Predicting the cost-effectiveness of regression testing strategies. In Proceedings of the ACMSIGSOFT '96 Fourth Symposium on the Foundations of Software Engineering, Oct. 1996.

9. H. Leung and L. White. A cost model to compare regression test strategies. In Proc. of Conf. on Software. Maint. Pages 201-208, Oct.1991.

10. Harsh Bhasin, Manoj. Regression Testing Using Coupling and GeneticAlgorithms.IN (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 3 (1), 2012, 3255 – 3259.

11. Automation Testing Tutorial: What is Automated Testing. [Електронний ресурс] – Режим доступу до ресурсу: https://www.guru99.com/automation-testing.html

12. Selenium Webdriver [Електронний ресурс] – Режим доступу до ресурсу: https://drive.google.com/file/d/0B7TBmsv_w76nQ1FfQVdyWW RXYk0/ view.

13. Welcome to Apache Maven [Електронний ресурс] – Режим доступу до ресурсу: https://maven.apache.org/.

14. Hypertext Transfer Protocol -- HTTP/1.1 [Електронний ресурс]. – Режим доступу: https://www.rfc-editor.org/rfc/rfc2616.

15. Simple Object Access Protocol (SOAP) 1.1 [Електронний ресурс]. – Режим доступу: https://www.w3.org/TR/soap/.

16. Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map [Електронний ресурс]. – Режим доступу: https://www.rfc-editor.org/rfc/rfc4510.

17. An all-in-one test automation solution [Електронний ресурс] – Режим доступу до ресурсу: https://www.katalon.com/.

18. Automation Testing Tutorial: What is Automated Testing. [Електронний ресурс] – Режим доступу до ресурсу: https://www.guru99.com/automation-testing.html.

19. Винниченко И. В. Автоматизация процессов тестирования / Винниченко И. В. – СПб. : Питер, 2011. – 203 с.

20. Top 10 Automation Testing Tools in 2021 [Електронний ресурс] – Режим доступу до ресурсу: https://www.netsolutions.com/insights/top-10-automation-testing-tools/.

21. Java [Електронний ресурс] – Режим доступу до ресурсу: https://uk.myservername.com/java-jdbc-tutorial-what-is-jdbc.

22. TCP [Електронний ресурс] – Режим доступу до ресурсу: http://elartu.tntu.edu.ua/bitstream/123456789/9607/2/Conf_2013v1_Radchuk_V-Printsip_roboti_protokolu_TCP_105.pdf.

23. Що таке CLI? [Електронний ресурс] – Режим доступу до ресурсу: https://uk.education-wiki.com/2019691-what-is-cli.

24. The Art of Application Performance Testing / Ian Molyneaux – M. : O'Reilly Media. – 2009. – №1. – 158 с.

25. Response time distribution and outlier analysis [Еклектронний ресурс] / dynatrace – режим доступу: www/URL:https://www.dynatrace.com/support/help/how-to-use-dynatrace /transactions-andservices/analysis /response-timedistribution-and-outlier -analysis/ − 17.11.2020 р. – назв. з екрану.

26. Integrated Approach to Web Performance Testing: A practitioner’s Guide / B.M. Subraya – M. Infosys Technologies Limited;, 2006. – 368 с.

27. A Study of Factors Affecting Websites Page Loading Speed for Efficient Web Performance / Jatinder Manhas/ International Journal of Computer Sciences and Engineering Vol. 1 No. 3, available at: https://www.researchgate.net /publication/274070742_A_Study_of_Factors_Affecting_Websites_Page_Loading _Speed_for_Efficient_Web_Performance.

28. L. Ljung, System Identification - Theory for the User, Prentice-Hall, Upper Saddle River, N.J., 2nd edition, 1999.

29. J. Sj¨oberg, Q. Zhang, L. Ljung, A. Benveniste, B. Delyon, P.Y. Glorennec, H. Hjalmarsson, and A. Juditsky, “Nonlinear black-box modeling in system identification: A unified overview,” Automatica, vol. 31, no. 12, pp. 1691–1724, 1995.

30. H. Akaike, “A new look at the statistical model identification,” IEEE Transactions on Automatic Control, vol. AC-19, pp. 716–723, 1974.

31. L.P. Wang and W. R. Cluett, “Frequency-sampling filters: An imporves model structure for step-response identification,” Automatica, vol. 33, no. 5, pp. 939–944, May 1997.

32. J. Schoukens and R. Pintelon, Identification of Linear Systems: A Practical Guideline to Accurate Modeling, Pergamon Press, London (U.K.), 1991.

33. Armax моделі [Електронний ресурс] – режим доступу: https://docs.exponenta.ru/ident/ref/armax.html.

34. Web server [Еклектронний ресурс] – режим доступу: https://books.google.com.ua/books?id=0jExRH3_-hQC&q="Web+server"+wikipedia&pg=PA14&redir_esc=y#v=snippet&q="Web%20server"%20-wikipedia&f=false

35. Коди відповіді стану HTTP [Електронний ресурс] – Режим доступу: https://sebweo.com/shpargalka-po-kodam-vidpovidi-stanu-http/.

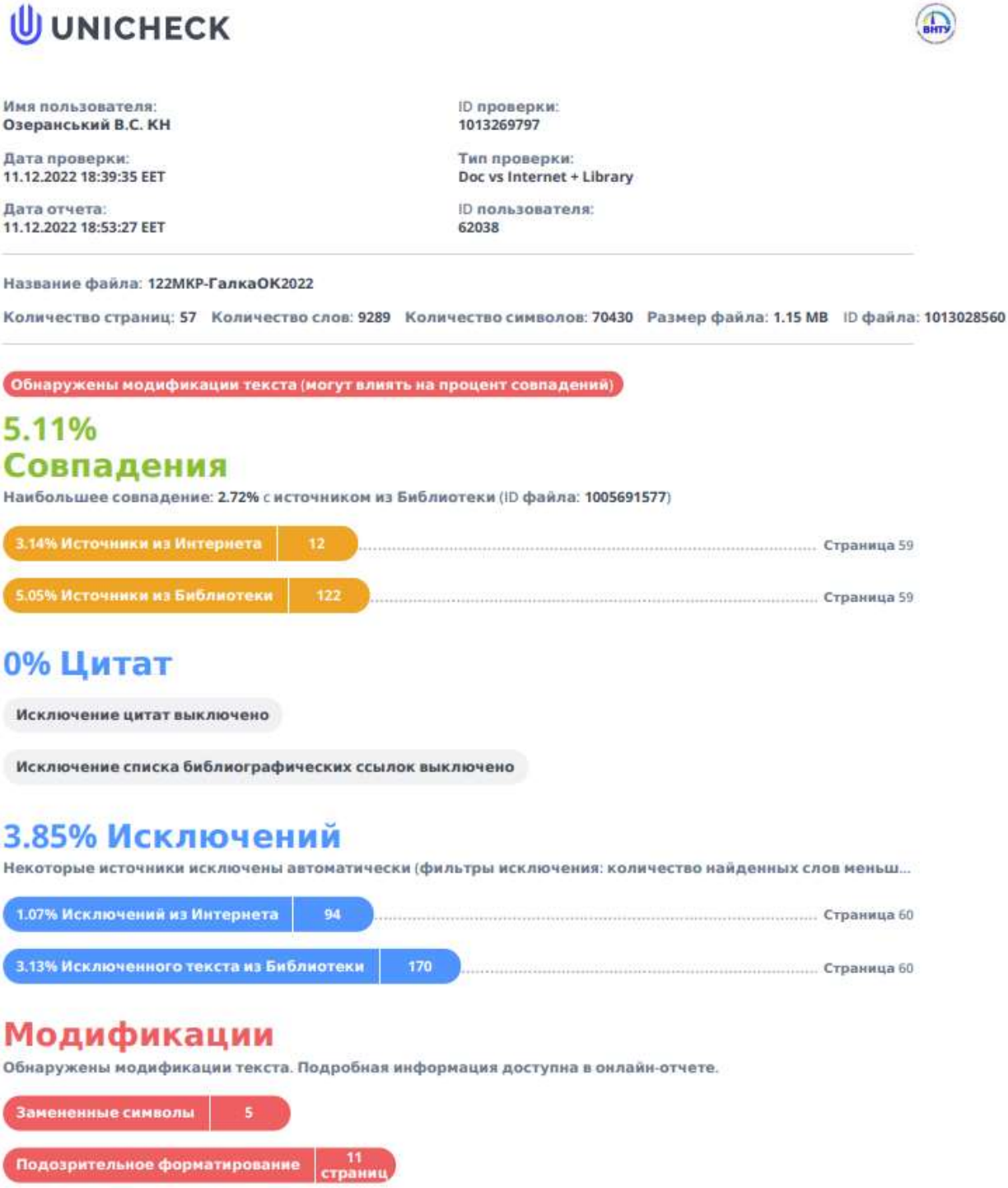
36. Розрахунок тарифів на електроенергію в навчальних роботах [Електронний ресурс] – Режим доступу: https://epvmvntu.wordpress.com/2021/09/16/

37. Цікаві факти про Python [Електронний ресурс] – Режим доступу: http://www.itschool.vn.ua/interesting-python/

38. About us [Електронний ресурс] - Режим доступу: https://www.raspberrypi.org/about/

39. Introducing Thonny, a Python IDE for learning programming [Електронний ресурс] – Режим доступу: https://dl.acm.org/doi/10.1145/2828959.2828969

Додаток А
(обов'язковий)
Результат перевірки на антиплагіат у системі «UNICHECK»



UNICHECK

Имя пользователя:
Озеранський В.С. КН

ID проверки:
1013269797

Дата проверки:
11.12.2022 18:39:35 EET

Тип проверки:
Doc vs Internet + Library

Дата отчета:
11.12.2022 18:53:27 EET

ID пользователя:
62038

Название файла: 122МКР-ГалкаОК2022

Количество страниц: 57 Количество слов: 9289 Количество символов: 70430 Размер файла: 1.15 MB ID файла: 1013028560

Обнаружены модификации текста (могут влиять на процент совпадений)

5.11%
Совпадения

Наибольшее совпадение: 2.72% с источником из Библиотеки (ID файла: 1005691577)

3.14% Источники из Интернета 12 ..... Страница 59

5.05% Источники из Библиотеки 122 ..... Страница 59

0% Цитат

Исключение цитат выключено

Исключение списка библиографических ссылок выключено

3.85% Исключений

Некоторые источники исключены автоматически (фильтры исключения: количество найденных слов меньш...

1.07% Исключений из Интернета 94 ..... Страница 60

3.13% Исключенного текста из Библиотеки 170 ..... Страница 60

Модификации

Обнаружены модификации текста. Подробная информация доступна в онлайн-отчете.

Замененные символы 5

Подозрительное форматирование 11 страниц

Додаток Б

(обов'язковий)

Лістинг програми

Лістинг програми для запису та редагування даних користувача

```
import datetime
import logging

import jwt
from bson import ObjectId
from flask import request, g
from flask_login import UserMixin as LoginUserMixin
from flask_user import UserMixin, UserManager, db_manager, PasswordManager
from itsdangerous import TimedJSONWebSignatureSerializer as Serializer, SignatureExpired, BadSignature
from pymongo.errors import PyMongoError

from core.config import app, db, oauth, encription
from core.helpers.static_tools import get_route_func_name
from core.models.BlacklistToken import BlacklistToken
from core.models.base.DbModelBase import DbModelBase
from core.tools import wrap_response
from errors.exceptions import AppBackendError, UserIdIsNotActiveError


class User(DbModelBase, db.Document, UserMixin, LoginUserMixin):
    email = db.EmailField(unique=True, max_length=255, required=True)
    password = db.StringField(max_length=255, required=True)
    registered_on = db.DateTimeField(default=datetime.datetime.now())
    is_active = db.BooleanField()
    is_admin = db.BooleanField(required=False)
    nickname = db.StringField(unique=True)
    status = db.StringField(default='used')
    meta = {'collection': 'users'}

    def __init__(self, email, password, nickname, *args, **kwargs):
        super(User, self).__init__(*args, **kwargs)
        self.email = email
        self.password = str(encription.encrypt(password.encode())).split("'")[1]
        self.registered_on = datetime.datetime.now()
        self.nickname = nickname
        self.is_active = True
        self.is_admin = False
        self.status = 'used'
```

```
@property
def decrypt_password(self):
    byte = str(encription.decrypt(self.password.encode())).split("'")[1].encode()
    return str(encription.decrypt(byte)).split("'")[1]

def verify_password(self, password):
    if password == self.decrypt_password:
        return True
    return False

@property
def schedules(self):
    return self.schedules.get_by_user_id(self.id)

@staticmethod
def encode_auth_token(user_id):
    payload = {
        'exp': datetime.datetime.utcnow() + datetime.timedelta(days=0, hours=2),
        'iat': datetime.datetime.utcnow(),
        'sub': str(user_id)
    }
    return jwt.encode(
        payload,
        app.config.get('SECRET_KEY'),
        algorithm='HS256'
    )

@staticmethod
def decode_auth_token(auth_token: str):
    try:
        payload = jwt.decode(
            auth_token,
            app.config.get('SECRET_KEY'),
            algorithms=['HS256']
        )
        is_blacklisted_token = BlacklistToken.check_blacklist(auth_token)
        if is_blacklisted_token:
            return 'Token blacklisted. Please log in again.'
        else:
            return ObjectId(payload['sub'])
```

```
except jwt.ExpiredSignatureError:
        return 'Signature expired. Please log in again.'
    except jwt.InvalidTokenError:
        return 'Invalid token. Please log in again.'

@staticmethod
def verify_auth_token(token):
    s = Serializer(app.config.get('SECRET_KEY'))
    try:
        data = s.loads(token)
    except SignatureExpired:
        return None
    except BadSignature:
        return None
    user = User.objects(id=data['id']).first()
    return user

@classmethod
def to_dict_list(cls, objs: list, **kwargs):
    result = []
    exclude_fields = ('password', 'schedule')
    for o in objs:
        row = {}
        for c in o._fields:
            if c not in exclude_fields:
                row[c] = getattr(o, c)
                if isinstance(row[c], ObjectId):
                    row[c] = str(row[c])
                if isinstance(row[c], db.Document):
                    row[c] = str(row[c].id)
        result.append(row)
    return result

def to_dict(self, **kwargs):
    row = {}
    exclude_fields = ('password', )
    for c in self._fields:
        if c not in exclude_fields:
            row[c] = getattr(self, c)
            if isinstance(row[c], ObjectId):
```

```
    row[c] = str(row[c])
            if isinstance(row[c], db.Document):
                row[c] = str(row[c].id)
        return row

    @classmethod
    def get_list(cls):
        """
        Get all users

        Returns:
            list[User]: Return list of users documents
        """
        # objs = [user for user in cls.objects(status='used')]
        objs = map(lambda user: user, cls.objects(status='used'))
        return objs

    @classmethod
    def upd_by_id(cls, rec_id, raw: dict):
        if raw.get('email', None):
            user = User.objects(email=raw.get('email')).first()
            if user:
                return {'errors': {'message': f'This email {raw.get("email")} is already in use'}}

        obj = User.get_by_id(rec_id)
        for r in raw:
            if hasattr(obj, r):
                if r == 'password':
                    setattr(obj, r, user_manager.password_manager.hash_password(raw[r]))
                else:
                    setattr(obj, r, raw[r])
        try:
            obj.save()
            return obj
        except PyMongoError as ex:
            obj.clear()
            raise AppBackendError(ex)

@app.before_request
def check_user():
```

```
if request.method != 'OPTIONS':
    route_func_name = None
    if request.endpoint:
        route_func_name = get_route_func_name(request.endpoint)
        if (route_func_name is not None and route_func_name in app.config['SKIP_LOGIN_LIST']) \
                or request.endpoint == 'static' \
                or 'flasgger' in request.endpoint:
            return
    auth_header = request.headers.get('Authorization')
    auth_token = None
    if auth_header:
        try:
            logging.info('Get token with bearer')
            auth_token = auth_header.split(' ')[1]
        except IndexError:
            response_object = {
                'message': 'Bearer token malformed'
            }
            return wrap_response({'errors': response_object}, True)
    if not auth_token:
        auth_token = oauth.access_token
        logging.info('Get token from swagger')
    if auth_token:
        resp = User.decode_auth_token(auth_token)
        if isinstance(resp, str):
            response_object = {
                'message': resp
            }
            return wrap_response({'errors': response_object}, auth='Unauthorized')
        else:
            g.user = User.objects(id=resp).first()

            if not g.user.is_active and route_func_name not in app.config['ALLOW_INACTIVE_USER_LIST']:
                raise UserIdIsNotActiveError(user_id=g.user.id)
    else:
        response_object = {
            'message': 'Provide a valid auth token.'
        }
        return wrap_response({'errors': response_object}, True, auth='Unauthorized')
```

```
# setup Flask-User
user_manager = UserManager(app, db, User)
db_manager = db_manager.DBManager(app, db, User)
password_manages = PasswordManager(app)
```

Лістинг програми для запису даних проведених тестів в БД

```
import datetime
import threading

import requests
from pymongo.errors import PyMongoError

from core.config import db, logger
from core.models.Reports import Report
from core.models.base.DbModelBase import DbModelBase
from errors.exceptions import AppBackendError
from core.liighthouse.lighthouse import Lighthouse as House


class LightHouse(db.Document, DbModelBase):
    """ LightHouse Document for storing lighthouse related details """
    captchaResult = db.StringField()
    kind = db.StringField()
    L_id = db.StringField()
    loadingExperience = db.DictField()
    originLoadingExperience = db.DictField(required=False, default={})
    lighthouseResult = db.DictField()
    analysisUTCTimestamp = db.DateTimeField()
    meta = {'collection': 'Lighthouse'}

    def __init__(self, captchaResult, kind, L_id, loadingExperience, originLoadingExperience, lighthouseResult,
                 analysisUTCTimestamp, *args, **kwargs):
        """
        Initialise new Lighthouse

        :param str captchaResult: Captcha Result
        :return: Initialised Lighthouse
        :rtype: LightHouse
        """
        super(LightHouse, self).__init__(*args, **kwargs)
        self.captchaResult = captchaResult
        self.kind = kind
        self.L_id = L_id
        self.loadingExperience = loadingExperience
        self.originLoadingExperience = originLoadingExperience
        self.lighthouseResult = lighthouseResult
        self.analysisUTCTimestamp = analysisUTCTimestamp

    def __repr__(self):
        return f'<Lighthouse(captchaResult: {self.captchaResult}, kind: {self.kind}, id: {self.L_id},' \
               f'loadingExperience: {self.loadingExperience}, originLoadingExperience: {self.originLoadingExperience},
' \
               f'lighthouseResult: {self.lighthouseResult}, analysisUTCTimestamp: {self.analysisUTCTimestamp})>'
```

```
@staticmethod
    def add_new(post_data: dict, schedule, logg: bool = False, schedule_id: str = ''):
        """Add new Lighthouse report to collection

        Args:
            post_data (dict): Data with report
            logg (bool): Logging on
            schedule_id (str): Schedule run id
            schedule (Schedules): cSchedule document

        Raises:
            AppBackendError: Error

        Returns:
            LightHouse: Object with short report
        """
        try:
            url = 'https://' + post_data.get('url')
            form_factor = post_data.get('form_factor', 'desktop')
            params = {'url': url, 'strategy': form_factor,
                      'utm_source': 'lh-chrome-ext',
                      'category': ['best-practices', 'seo', 'pwa', 'performance', 'accessibility'],
                      }

            data = requests.get(
                'https://www.googleapis.com/pagespeedonline/v5/runPagespeed', params=params)
            dt = data.json()
            if data.status_code != 200:
                runner = House(url, form_factor)
                response_object = {'errors': {'message': data.reason}}
                # originLoadingExperience = runner.report.originLoadingExperience
                data = {
                    'metrics': {
                        'first_contentful_paint': [metric.score for metric in runner.report.metrics
                                                   if metric.id == 'first-contentful-paint'],
                        'speed_index': [metric.score for metric in runner.report.metrics
                                        if metric.id == 'speed-index'],
                        'interactive': [metric.score for metric in runner.report.metrics
                                        if metric.id == 'interactive'],
                        'largest_contentful_paint': [metric.score for metric in runner.report.metrics
                                                     if metric.id == 'largest-contentful-paint'],
                        'total_blocking_time': [metric.score for metric in runner.report.metrics
                                                if metric.id == 'total-blocking-time'],
                        'cumulative_layout_shift': [metric.score for metric in runner.report.metrics
                                                    if metric.id == 'cumulative-layout-shift'],
                        'first_meaningful_paint': [metric.score for metric in runner.report.metrics
                                                   if metric.id == 'first-meaningful-paint'],
                    },
                    'score': {
                        'performance': [metric.score for metric in runner.report.score
                                        if metric.id == 'performance'],
                        'accessibility': [metric.score for metric in runner.report.score
                                          if metric.id == 'accessibility'],
                        'best-practices': [metric.score for metric in runner.report.score
                                           if metric.id == 'best-practices'],
                        'seo': [metric.score for metric in runner.report.score
                                if metric.id == 'seo'],
                        'pwa': [metric.score for metric in runner.report.score
                                if metric.id == 'pwa']
                    },
                    'url': url,
```

```
'image': [metric.details.get('data') for metric in runner.report.metrics
                        if metric.id == 'final-screenshot'],
                'formFactor': form_factor
            }
            obj = LightHouse(
                captchaResult='CAPTCHA_NOT_NEEDED',
                kind='pagespeedonline#result',
                L_id=url,
                loadingExperience={},
                originLoadingExperience={},
                lighthouseResult={},
                analysisUTCTimestamp=datetime.datetime.utcnow()
            ).save()
            report = Report.add_new(data, obj)
            logger.warn(f'Added new Lighthouse: {obj.id}')
            logger.error(response_object)
            return report.to_dict()
        else:
            originLoadingExperience = dt.get('originLoadingExperience')
            obj = LightHouse(
                captchaResult=dt.get('captchaResult'),
                kind=dt.get('kind'),
                L_id=dt.get('id'),
                loadingExperience=dt.get('loadingExperience'),
                originLoadingExperience=originLoadingExperience if originLoadingExperience else {},
                lighthouseResult=dt.get('lighthouseResult'),
                analysisUTCTimestamp=datetime.datetime.utcnow()
            ).save()
            logger.warn(f'Added new Lighthouse: {obj.id}')
            result = obj.get_short_report(str(obj.id))
            if logg:
                print(f'Schedule with id = {schedule_id} complete. {threading.currentThread().getName()}')
                schedule.is_active = False
                schedule.save()
                data = Report.add_new(result, obj)
                schedule.update(push__report=data)
                return data
            data = Report.add_new(result, obj).to_dict()
            return data
    except PyMongoError as ex:
        raise AppBackendError(ex)

@classmethod
def get_short_report(cls, rec_id: str):
    """Get short report

    Args:
        rec_id (str): ObjectId

    Raises:
        AppBackendError: Error

    Returns:
        dict[str, dict[str, Any] | dict[str, Any]]: Generate short report
    """
    try:
        obj = cls.get_by_id(rec_id)
        raw = {}
        for c in obj._fields:
            if c == 'lighthouseResult':
                for report in getattr(obj, c):
```

```
if report == 'audits':
                    metrics = {
                      'first_contentful_paint': obj.lighthouseResult[report]['first-contentful-paint'][
                        'numericValue'],
                      'speed_index': obj.lighthouseResult[report]['speed-index']['numericValue'],
                      'interactive': obj.lighthouseResult[report]['interactive']['numericValue'],
                      'largest_contentful_paint': obj.lighthouseResult[report]['largest-contentful-paint'][
                        'numericValue'],
                      'total_blocking_time': obj.lighthouseResult[report]['total-blocking-time'][
                        'numericValue'],
                      'cumulative_layout_shift': obj.lighthouseResult[report]['cumulative-layout-shift'][
                        'numericValue'],
                      'first_meaningful_paint': obj.lighthouseResult[report]['first-meaningful-paint'][
                        'numericValue'],
                    }

                    raw.update(
                      {'image': obj.lighthouseResult[report]['final-screenshot']['details']['data']})
                    raw.update({'metrics': metrics})
                  if report == 'categories':
                    score = {
                      'performance': obj.lighthouseResult[report]['performance']['score'],
                      'accessibility': obj.lighthouseResult[report]['accessibility']['score'],
                      'best-practices': obj.lighthouseResult[report]['best-practices']['score'],
                      'seo': obj.lighthouseResult[report]['seo']['score'],
                      'pwa': obj.lighthouseResult[report]['pwa']['score'],
                    }
                    raw.update({'score': score})
                  if report == 'requestedUrl':
                    raw.update({'url': obj.lighthouseResult[report]})
                  if report == 'configSettings':
                    raw.update(
                      {'formFactor': obj.lighthouseResult[report]['formFactor']})
      except PyMongoError as ex:
        raise AppBackendError(ex)

      return raw
```

Додаток В
(обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ WEB-ДОДАТКІВ. СЕРВЕРНА ЧАСТИНА

Рис. В.1 - Схема класифікації тестування за принципом роботи

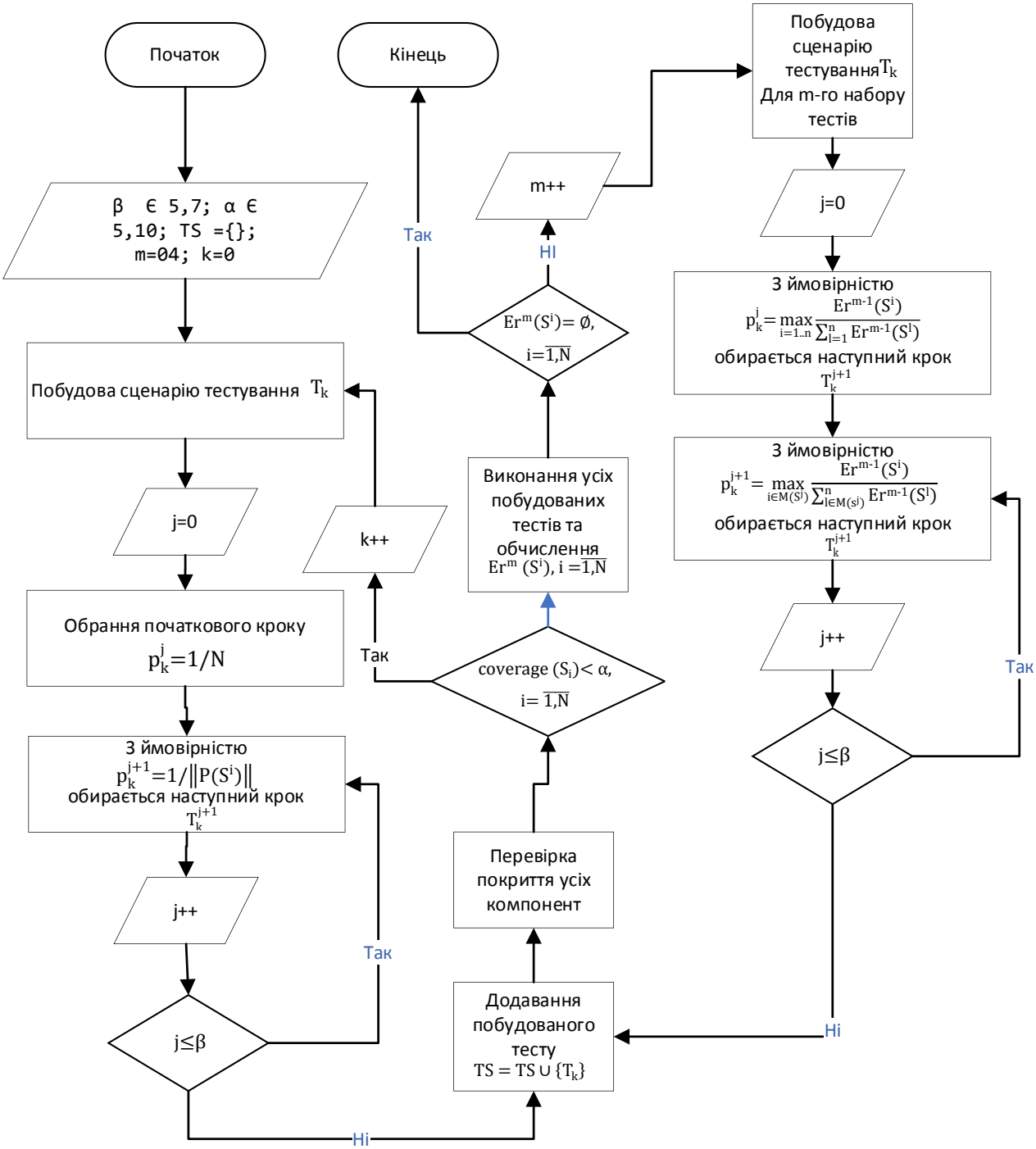


Рис. В.2 - Схема алгоритму автоматизованої побудови тестових сценаріїв

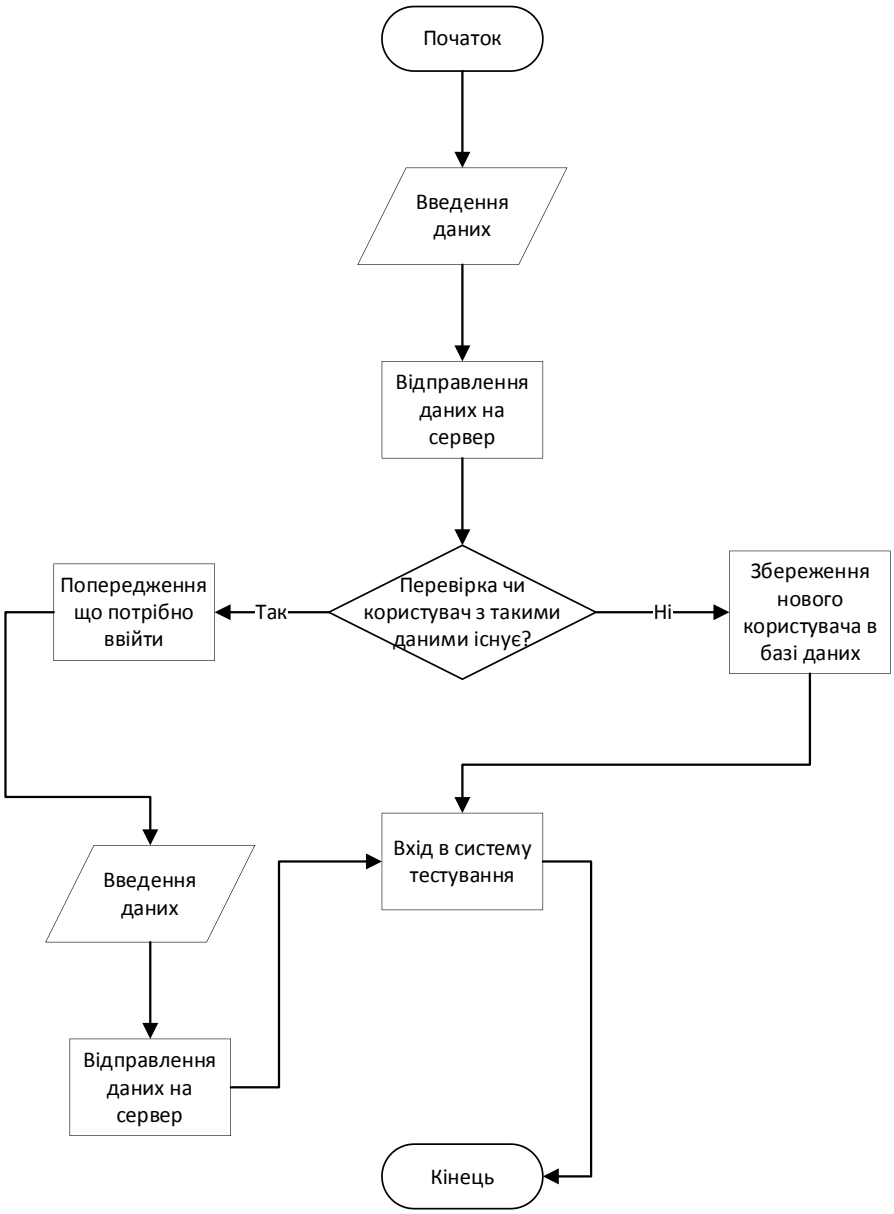


Рис. В.3 - Схема алгоритму реєстрації та входу користувача в систему тестувань

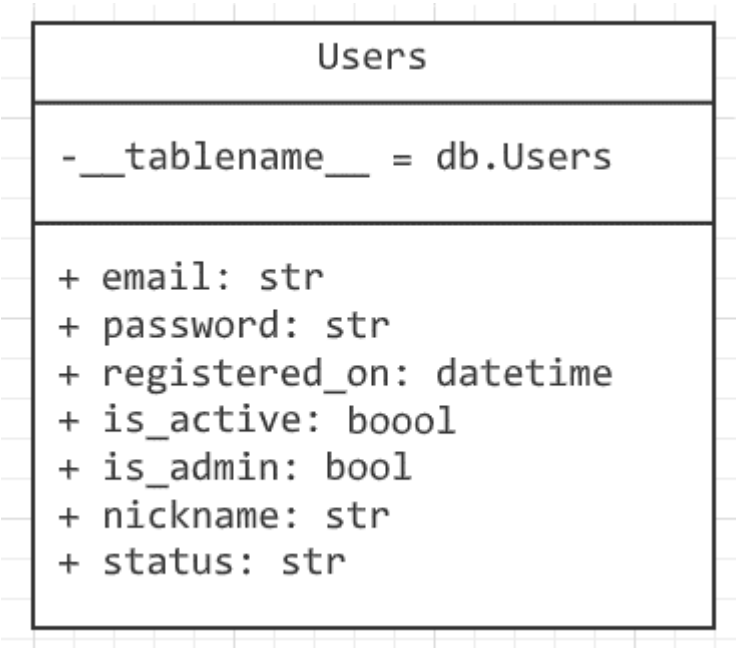


Рис. В.4 - Діаграма класу «Користувача»

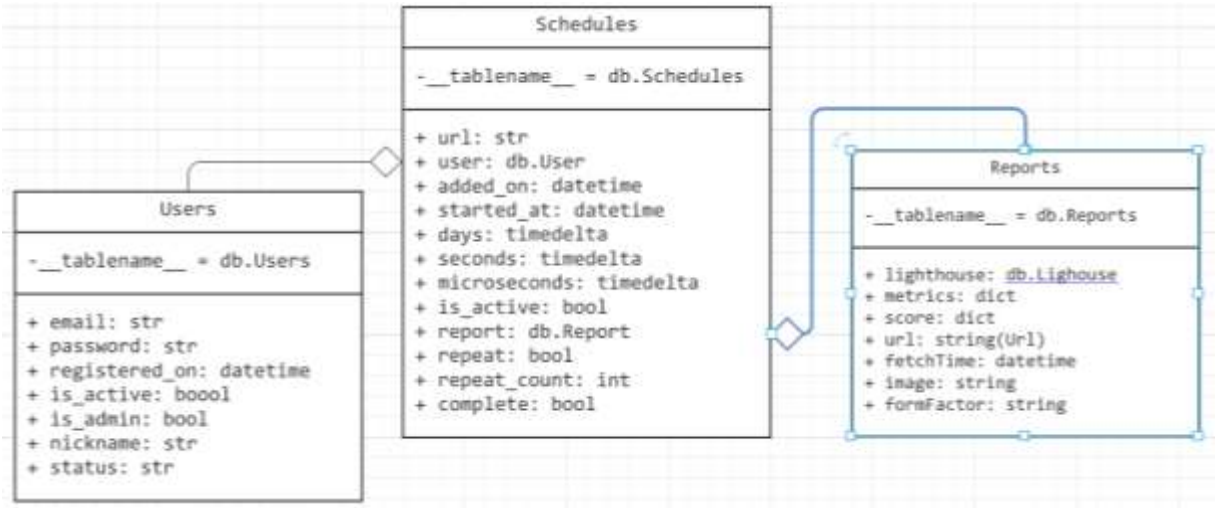


Рис. В.5 - Діаграма класів Schedules, Users, Reports

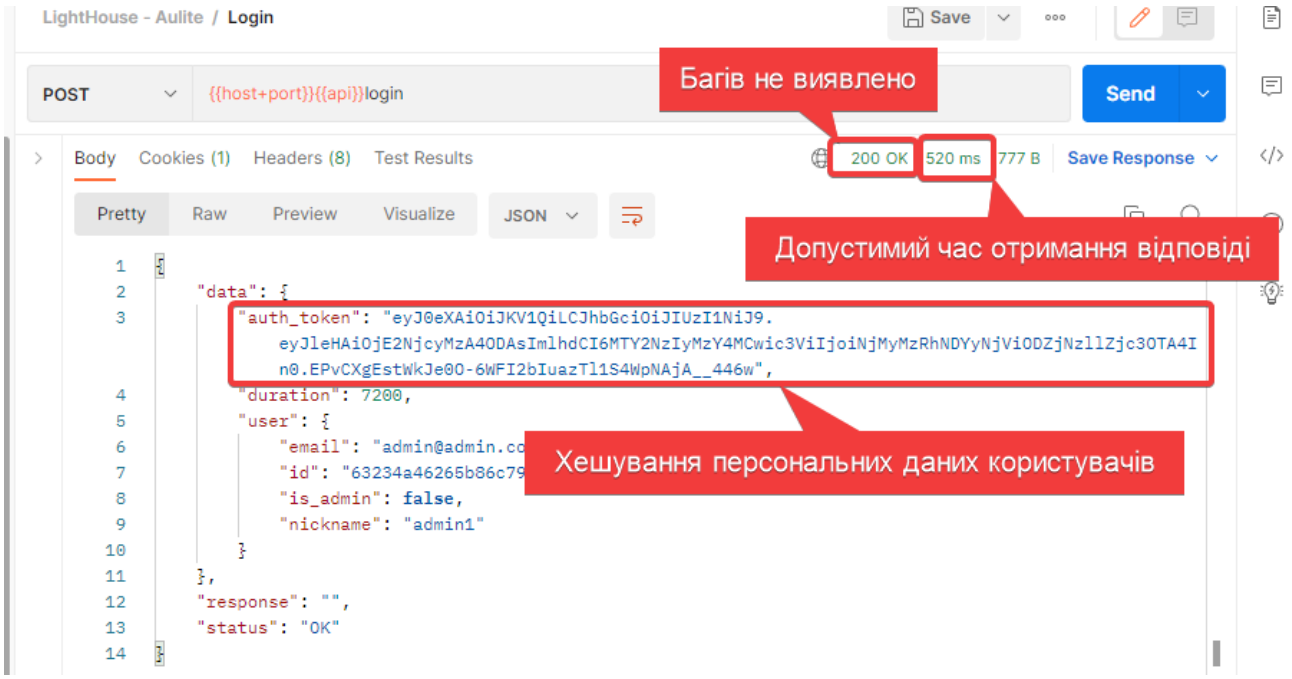


Рис. В.6 - Загальний вигляд інтерфейсного вікна із показниками надійності серверної системи

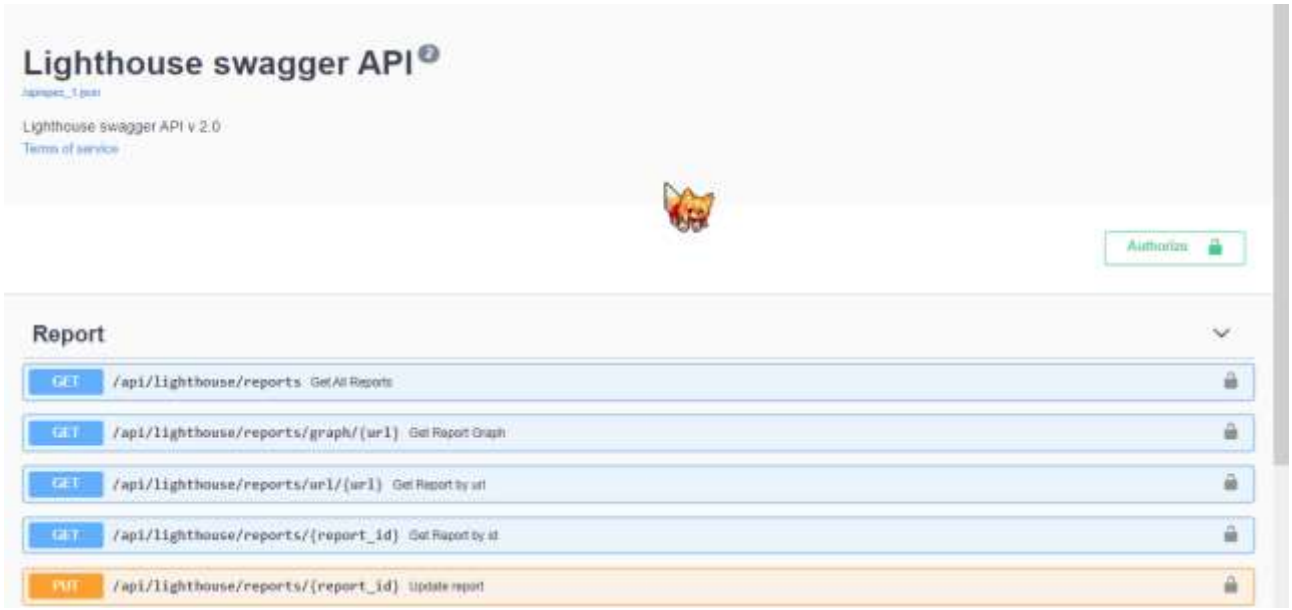


Рис. В.7 - Загальний вигляд головного меню інтерфейсу сервера

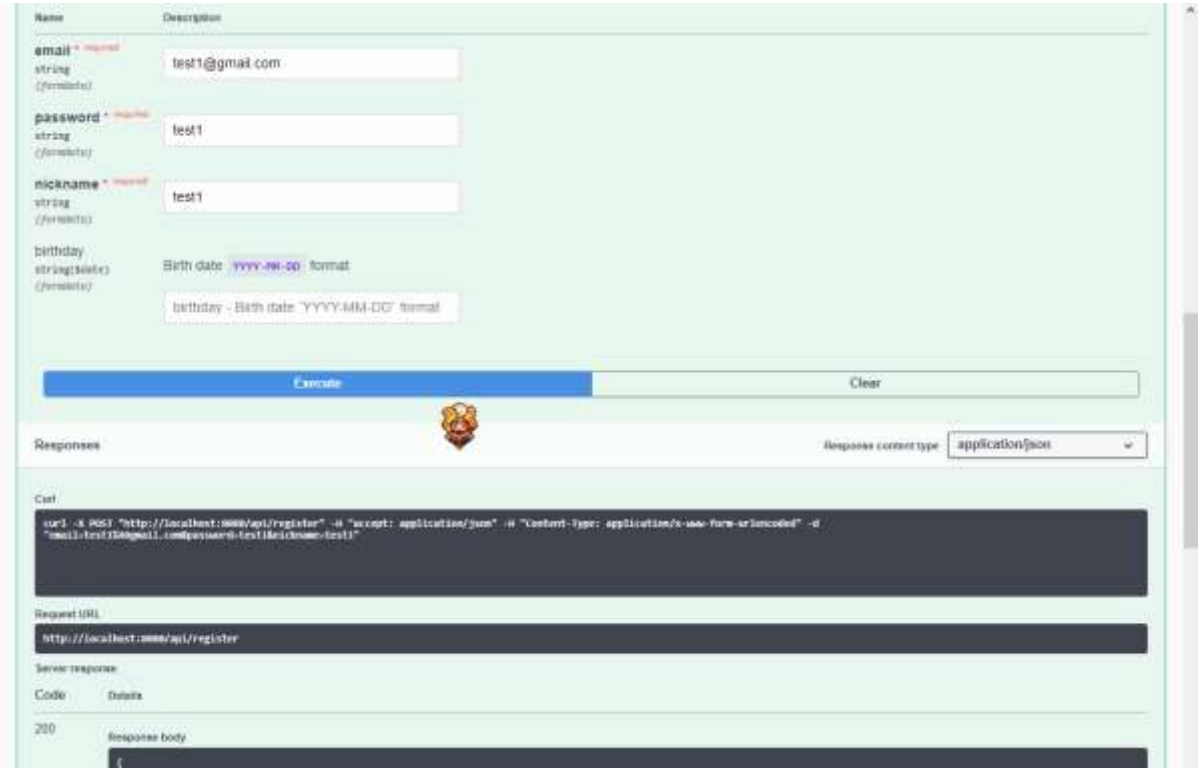


Рис. В.8 - Загальний вигляд інтерфейсу роботи реєстрації

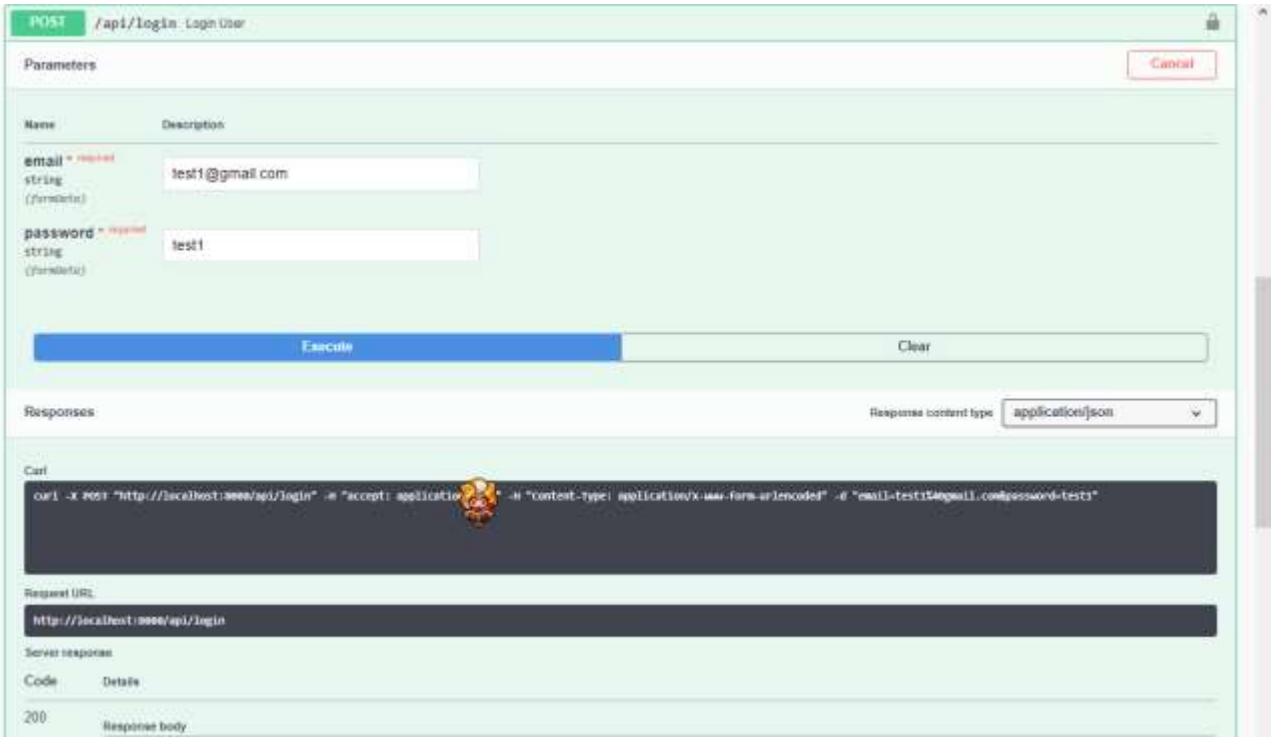


Рис. В.9 - Загальний вигляд інтерфейсу роботи входу в систему

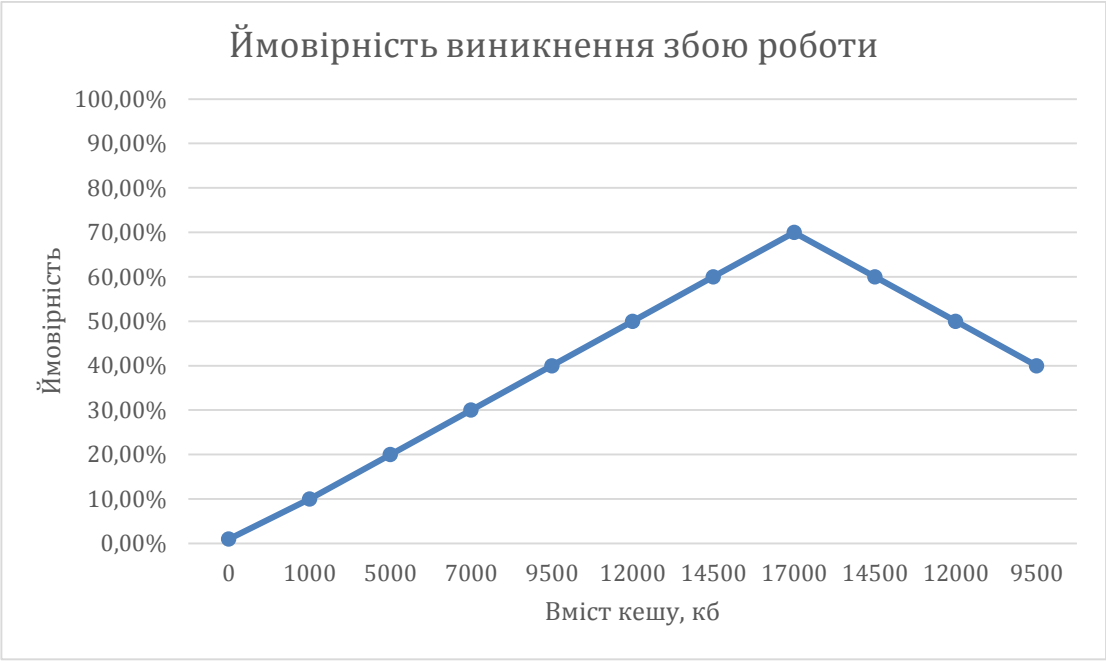


Рисунок В.10 – Діаграма ймовірності виникнення збою роботи програмного модуля серверної частини від вмісту хешованих даних на сервері

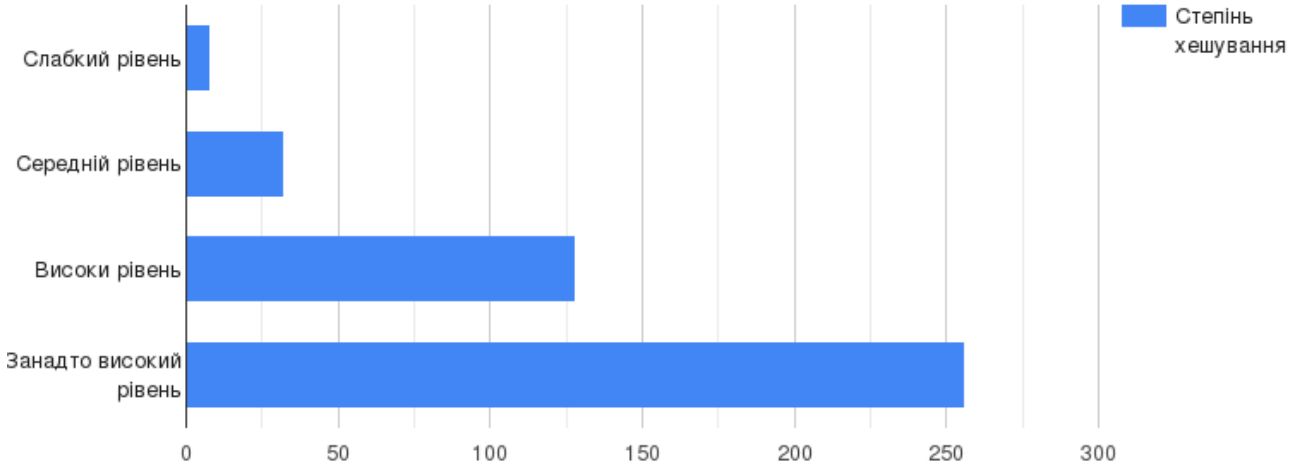


Рисунок В.11– Діаграма залежності рівня безпеки персональних даних в степені хешування

Додаток Г

(обов'язковий)

Інструкція користувача

Після того як сервер завантажився, переходим за адресою http://127.0.0.1:8080 і показується сторінка прикладного програмного інтерфейсу (API).

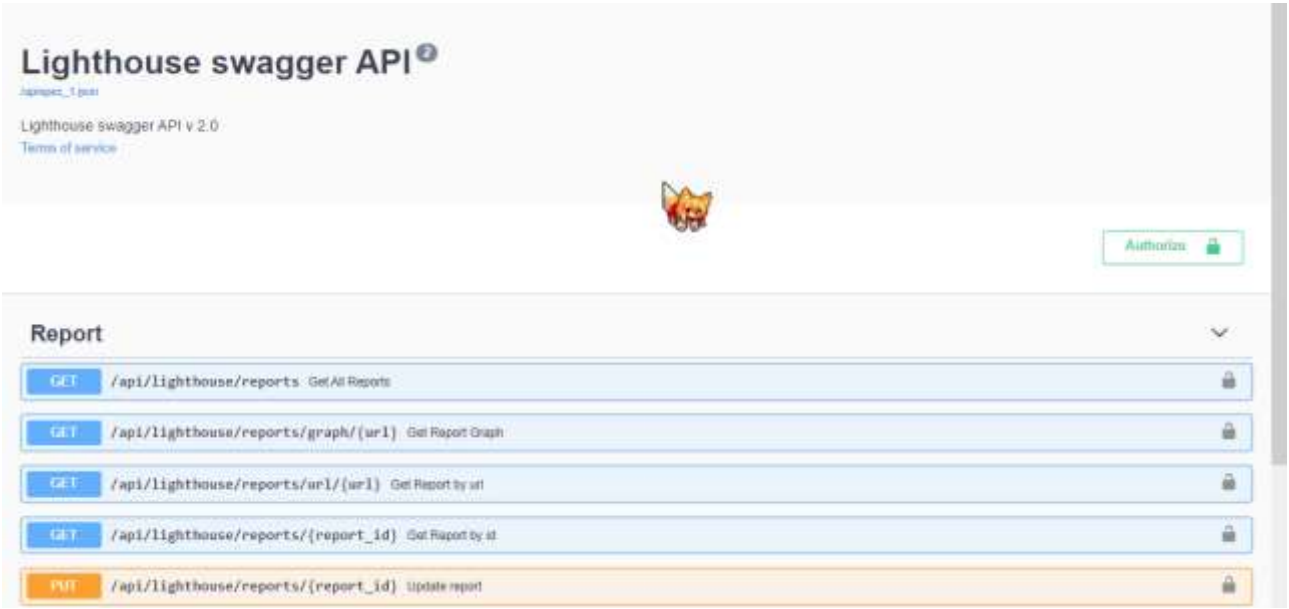


Рисунок Г.1 – Загальний вигляд прикладного програмного інтерфейсу інформаційної технології

Далі провіримо тестування системи реєстрації та авторизації на API зображено на рисунку Г.2-Г.3:

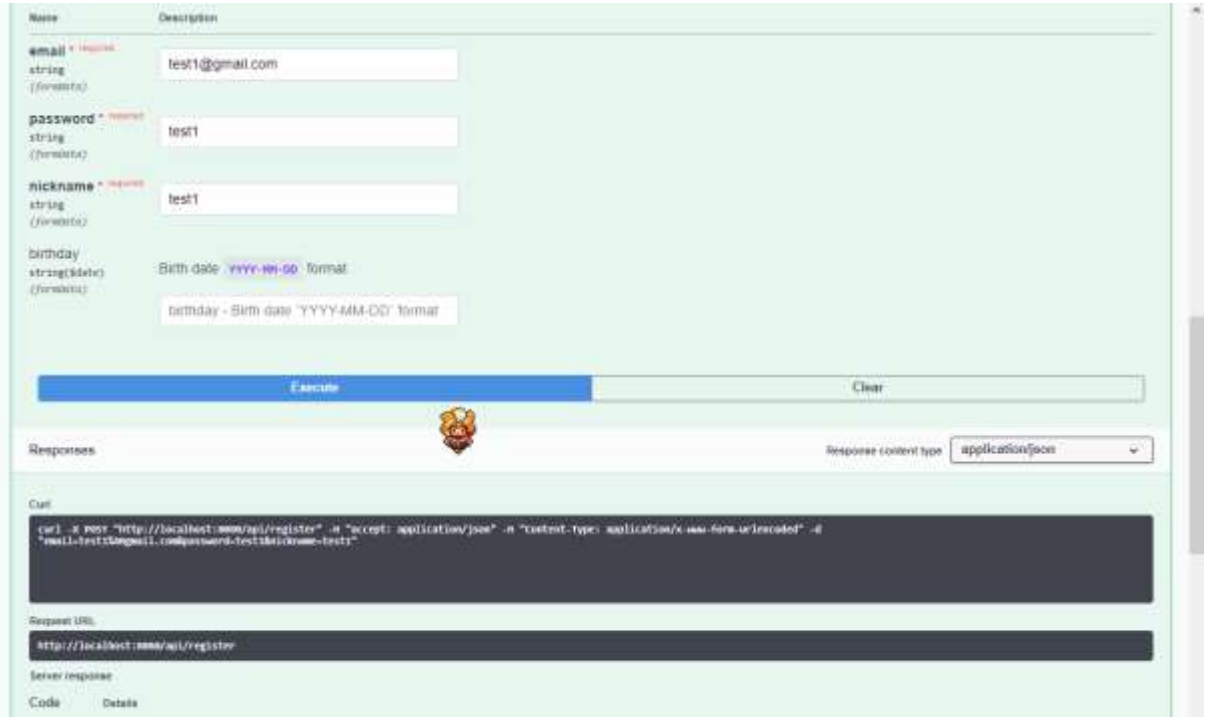


Рисунок Г.2 – Загальний вигляд інтерфейсу ідентифікації користувача за допомогою API (/api/register)

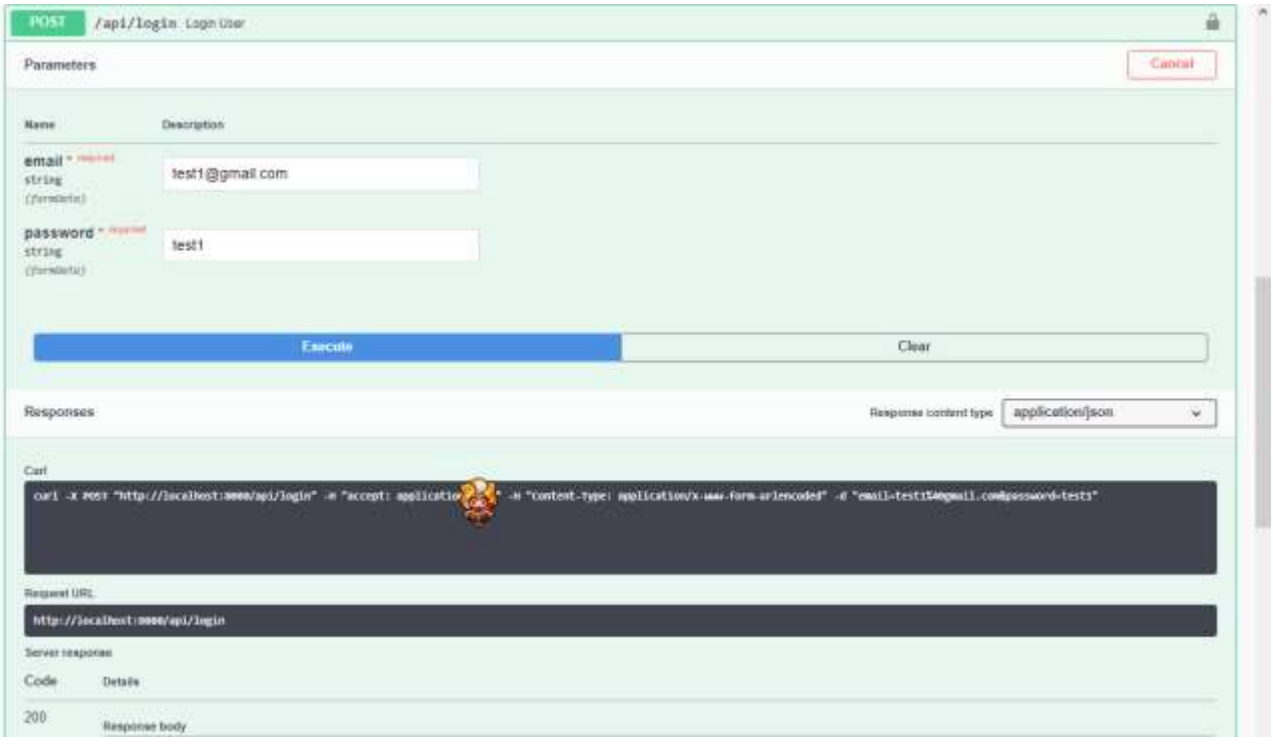


Рисунок Г.3 – Загальний вигляд інтерфейсу ідентифікації користувача за допомогою API(/api/login)

Якщо користувач не зареєстрований в системі, переходимо в розділ User Authentication.

Для реєстрації користувача потрібно заповнити поля «Email, Password, Nickname», після чого натискаєм кнопку “Execute” який відправляє запит і якщо користувач з такою поштою не зареєстрований тоді реєстрація проходить успішно, після чого потрібно авторизуватись.

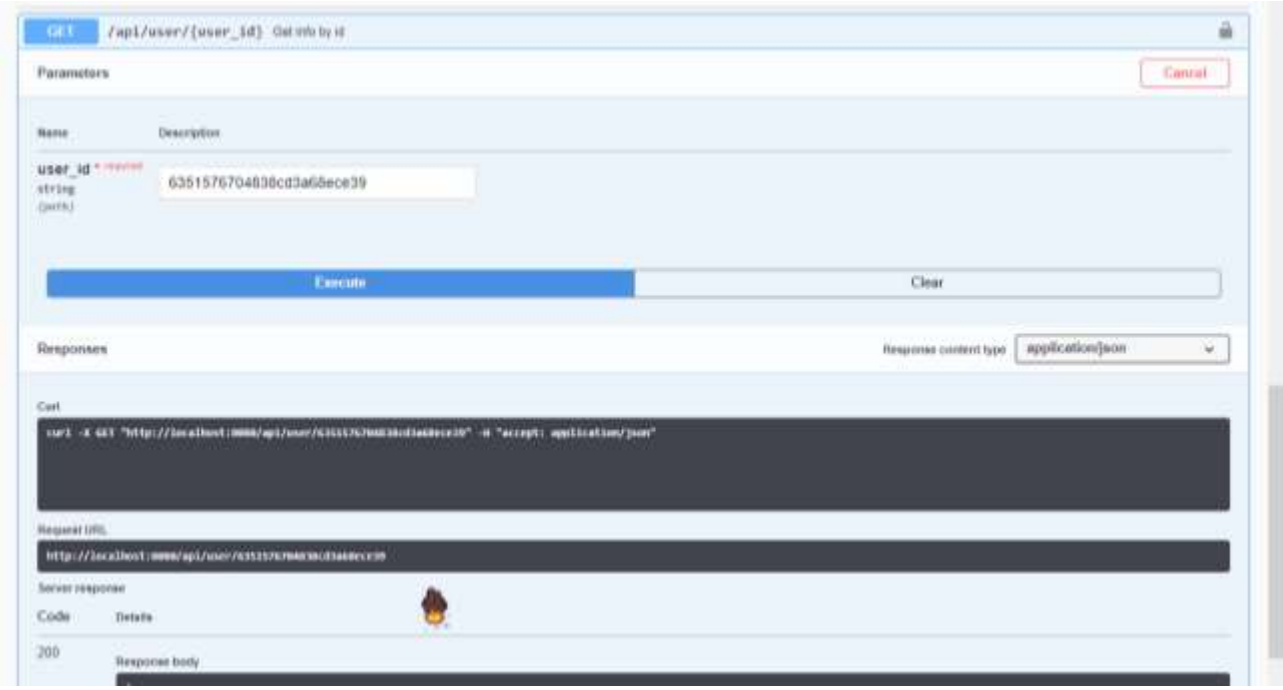


Рисунок Г.4 – Загальний вигляд інтерфейсу даних користувача за допомогою API(/api/user/<user_id>)

Протестуємо захист від неавторизованих користувачів. Він не дає змогу редагувати, створювати дані в базі даних. Як видно нижче на рисунку Г.5 при відправці даних неавторизованим користувачем спливає помилковий запит 401

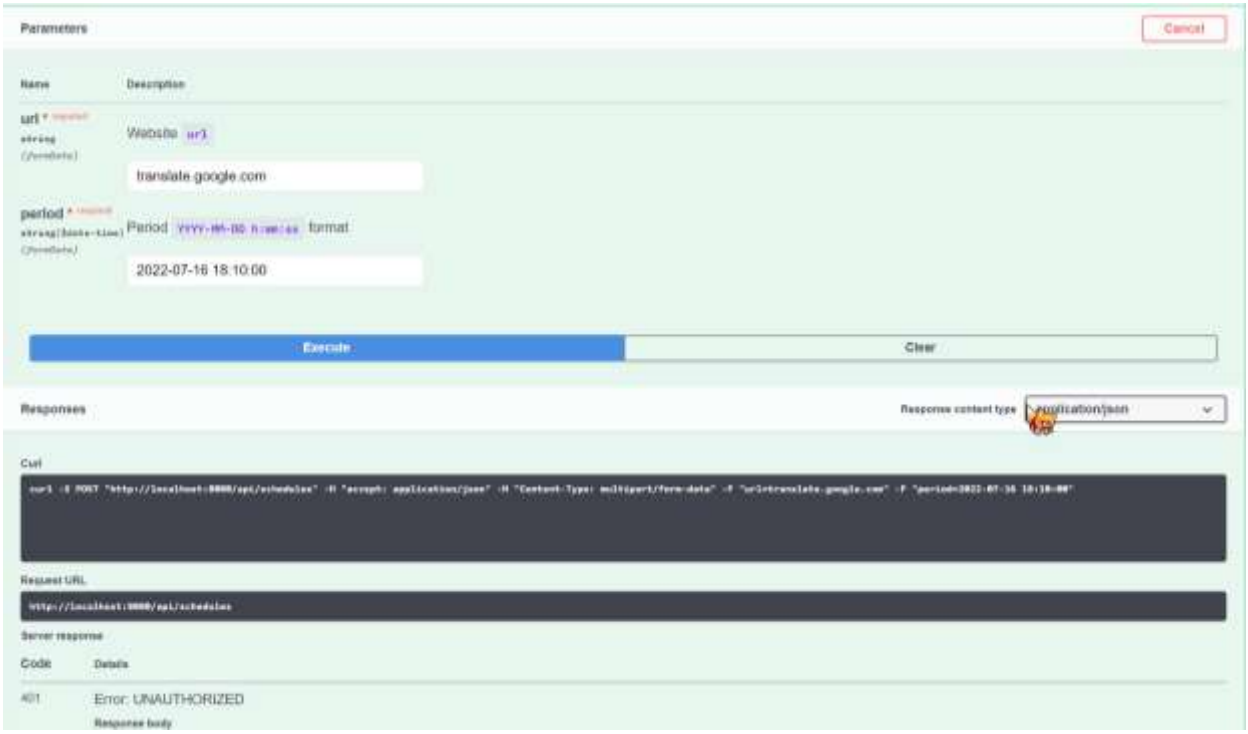


Рисунок Г.5 – Загальний вигляд захисту від неавторизованих користувачів