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. Клієнтська частина»

Виконав: студент 2-го курсу, групи 2КН-21м
спеціальності 122 – Комп'ютерні науки
Затковський В. Р.
(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)
«15» грудня 2022 р.

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

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти II-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

Затковському Віталію Романовичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Інформаційна технологія автоматизованого тестування WEB-додатків. Клієнтська 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 до роботи: зручний інтерфейс для взаємодії користувача з системою, мова програмування – наявність об'єктно-орієнтованого програмування (ООП); час початкового відображення вмісту 1000 мс; час завантаження WEB-сторінки 1300 мс; час відображення найбільшої частини вмісту 1200 мс; час до взаємодії зі сторінкою 2500 мс; загальний час блокування WEB-сторінки 150 мс; кумулятивний зсув макета (CLS) 0,10 ум. од.
4. Зміст текстової частини: вступ, аналіз сучасного стану розвитку систем автоматизованого тестування WEB-додатків, проектування інтелектуальної моделі системи автоматизованого тестування WEB-додатків, програмна реалізація клієнтської частини системи автоматизованого тестування WEB-додатків, висновок, перелік використаної літератури.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень): схема архітектури динамічного тестування, схема проведення динамічних тестів, загальна структурна схема клієнтської частини системи автоматизованого тестування WEB-додатків, алгоритм роботи розробленого методу автоматизованого тестування WEB-додатків, схема інтерфейсу головної сторінки системи автоматизованого тестування WEB-додатків, загальний вигляд головної сторінки програного модуля клієнтської частини автоматизованого тестування WEB-додатків із заповненим розкладом, загальний вигляд прикладу звіту проведеного тестування у інформаційній технології автоматизованого тестування WEB-додатків.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийн
Спеціальна частина. Розділ 1-3.	Іванчук Я.В., д.т.н., проф. каф. КН	14.09.2022	14.09.2022
Економічна частина	Бурєнникова Н.В., професор каф. ЕПВМ	14.09.2022	14.09.2022

7. Дата видачі завдання 14.09 2022 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Пр мі
1	Аналіз сучасного стану розвитку систем автоматизованого тестування WEB-додатків	Аналітичний огляд літературних джерел, задачі досліджень, розділ 1 14.09-01.10	
2	Моделювання процесу роботи системи	Моделі, розділ 2 02.10-16.10	
3	Розробка клієнтської системи автоматичного тестування WEB-додатків	Розділ 3 17.10 - 07.11	
4	Програмна реалізація системи автоматичного тестування WEB-додатків	Розділ 3 08.11.-21.11	
5	Апробація та/або впровадження результатів дослідження	Тези доповідей/акт впровадження 23.11-01.12	
6	Оформлення пояснювальної записки, графічного матеріалу та презентації	Пояснювальна записка, графічний матеріал, 02.12-презентація 14.12	

Студент _____ (підпис) Затковський В. Р.

Керівник роботи _____ (підпис) Іванчук Я. В.

АНОТАЦІЯ

УДК 004.054

Затковський В. Р. Інформаційна технологія автоматизованого тестування WEB-додатків. Клієнтська 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 динамічного тестування. Також розроблено метод автоматизації динамічного тестування. Розроблена клієнтська частина, призначена для автоматизованого тестування. Для розробки проекту використано програмне середовище VS Code, мова програмування TypeScript та JavaScript. Для розробки одно-сторінкового застосунку було використано бібліотеку React, для збереження даних використовувалась об’єктно-орієнтована база даних MongoDB. Визначено, що при використанні даної технології, результати ефективності збільшуються у щонайменше 1,32 рази. Графічна частина складається з 7 плакатів із результатами проектування та реалізації.

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

ABSTRACT

UDC 004.054

Zatkovskyi V. R. Information technology of automated testing of WEB applications. Client 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 dynamic testing was developed. A method of automating dynamic testing has also been developed. The client part designed for automated testing has been developed. The VS Code programming environment, the TypeScript and JavaScript programming languages were used to develop the project. The React library was used to develop a one-page application, and the MongoDB object-oriented database was used to store data. It was determined that when using this technology, the efficiency results increase by at least 1,32 times. The graphic part consists of 7 posters with the results of design and implementation.

Keywords: productivity; automation; testing; web application.

ЗМІСТ

ВСТУП

У процесі розробки WEB-додатків неминуче наявність помилок. Деякі дрібні помилки залишаються непоміченими, але інші можуть загрожувати стабільній роботі програми або фінансовим витратам. Тому вкрай важливо тестувати програмні продукти, щоб переконатися, що помилки не потрапили до релізу [1].

Тестування програмних продуктів є ефективним способом заощадження коштів завдяки своїй економічній ефективності. Виправлення помилок на попередніх етапах розробки програмного забезпечення або WEB-додатків є не таким дорогим, на відмінну від їх пізнішого виправлення. Ось чому важливо досліджувати та вирішувати будь-які проблеми як найшвидше [1, 2].

Питання, які відносяться до тестування WEB-додатків, є динамічними та актуальними для поточного стану справ. Це пов’язано з тим, що WEB-програми пронизують усі аспекти людської діяльності. Тестування продуктивності має гарантувати правильну роботу WEB-додатків незалежно від того, що спричиняє виконання тесту: низька пропускна здатність, інтенсивний трафік або непостійний мережевий трафік [1].

Автоматизація є одним з напрямів науково-технічного прогресу, який спрямовано на застосування саморегульованих технічних засобів, методів і систем керування, що звільняють людину від участі у процесах отримання, перетворення, передавання і використання ресурсів, що істотно зменшують міру участі чи трудомісткість виконуваних операцій [2].

Основною перевагою автоматизації тестування є швидке, автоматичне виконання набору тестів, що дозволяє оперативно вносити зміни у програмний код WEB-додатку для коригування відповідних недоліків виявлених на етапі коригування [4]. Крім того, сучасні WEB-додатки використовують багаторівневу

архітектуру, що дозволяє реалізувати програмний код шляхом розподілення потужностей комп'ютерних систем. З цієї причини, для перевірки стійкості і коректності роботи WEB-додатків, необхідно використовувати наскрізні методи тестування [4].

Актуальність автоматизації тестування при розробці WEB-додатків зумовлена економією часу на етапі проектування, а також зменшенням кількості помилок програмного коду на завершальному етапі виконання проекту [3]. Тому актуальним є розробка інформаційної технології автоматизованого тестування WEB-додатків, яка дозволить автоматизувати процес тестування для підвищення рівня працездатності WEB-додатків.

Метою магістерської кваліфікаційної роботи є підвищення рівня відповідності вимогам забезпечення якості програмного забезпечення за допомогою розробки інформаційної технології автоматизованого тестування WEB-додатків на клієнтському рівні.

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

- провести аналіз предметної області автоматизованого тестування WEB-додатків;
- провести аналіз відомих технічних рішень клієнтських 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 – клієнтські програмні засоби автоматизації тестування WEB-додатків.

Методи дослідження – методи та засоби розробки автоматизованих систем WEB-орієнтованих програмних додатків, методи генерації динамічних тестів, методи проведення тестувань, засоби обробки інформації.

Наукова новизна одержаних результатів полягає в наступному:

Розроблено інформаційну технологію автоматизованого тестування WEB-додатків, що відрізняється від існуючих наявністю методу проведення динамічних тестів на основі розробленої математичної моделі ідентифікації степеню продуктивності WEB-додатків, що дозволяє автоматизувати процес тестування для підвищення рівня працездатності WEB-додатків.

Практичне значення одержаних результатів полягає у наступному:

1. Розроблено структурну схему архітектури клієнтської частини системи автоматизованого тестування WEB-додатків.

2. Розроблено програмний модуль клієнтської частини системи автоматизованого тестування WEB-додатків, що відрізняється від існуючих, використанням моделі з підвищеною ефективністю перевірки відповідності вимогам забезпечення якості програмного забезпечення 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 ТЕСТУВАННЯ WEB-ДОДАТКІВ

1.1 Аналіз предметної області автоматизованого тестування WEB-додатків

WEB-додатки відіграють важливу роль в житті нашого суспільства. Вони застосовуються в таких секторах, як бізнес, охорона здоров'я та державне управління. Від якості таких додатків може залежати не лише зручність користувачів, але і функціонування організацій. Тестування є найбільш широко використовуваним і ефективним підходом для забезпечення якості та надійності програмного забезпечення, включаючи WEB-додатки. Проте, WEB-додатки дуже відрізняються від традиційного програмного забезпечення, оскільки вони включають в себе динамічне створення та інтерпретацію коду, а також реалізацію конкретного режиму взаємодії на основі навігаційної структури WEB-програми [2].

У випадку ручного тестування, тестер бере на себе роль кінцевого користувача, перевіряючи WEB-додаток для заподіяти несподівану поведінку чи помилку. Для професійної діагностики тестувальник часто користується написаним планом тестування з варіантами тестування – тест-кейсами [3].

Тестування – це трудомістка операція розробки застосунків. Це відомо принаймні як упередження, і відповідно в деяких проектах не проводиться багато тестування. Проте кожен, хто коли-небудь працював у проекті з хорошим тестовим покриттям, також знає, як це приємно бачити, як тести запускаються автоматично після впровадження нового компонента програми [2].

Автоматизоване тестування – це автоматичне виконання набору тестів. Створивши цей набір один раз, його можна використовувати кожного разу після

внесення деяких змін у WEB-додаток. Крім того, сучасні WEB-додатки побудовані на основі багаторівневої архітектури. Тому, щоб перевірити загальну поведінку WEB-додатків, потрібно скласти комплекс методів тестування. Автоматизація тестування не може бути реалізована без відповідних інструментів. Саме вони визначають, як буде здійснюватися тестування та чи можуть бути досягнуті переваги автоматизації. Інструменти автоматизації тестування є найважливішим компонентом у інструментальному ланцюжку розробки [4].

Важливість автоматизації тестування у WEB-розробках походить від широкого використання WEB-додатків та пов’язаної з нею потреби у якості коду. Автоматизація тестування вважається вирішальною для забезпечення рівня якості, що очікується користувачами, оскільки вона може заощадити багато часу в тестуванні, а також допомагає розробникам випускати WEB-програми з меншою кількістю дефектів. Основною перевагою автоматизації тестування є швидке, автоматичне виконання набору тестів після внесення деяких змін у WEB-додаток. Крім того, сучасні WEB-додатки використовують багаторівневу архітектуру, де реалізація розподіляється по різних шарах і запускається на різних машинах. З цієї причини, щоб перевірити загальну поведінку WEB-додатків, потрібні наскрізні методи тестування [5].

WEB-додаток (WEB-застосунок) – розподілений застосунок, в якому клієнтом виступає браузер, а сервером – WEB-сервер [2]. Браузер може бути представлений реалізацією так званих тонких клієнтів – логіка застосунку зосереджується на сервері, а браузер виконує функцію зображення інформації, завантаженої мережею із сервера, і передачі даних користувача. Однією з переваг такого підходу є незалежність клієнтів від типу операційної системи користувача, а це у свою чергу реалізує у WEB-застосунках функцію міжплатформеного сервісу [2, 4]. Унаслідок цієї універсальності й відносної простоти системної архітектури WEB-застосунки отримали широку популярність.

Тестування – це одна із технік контролю якості, що включає в себе планування, проєктування, виконання тестування та аналіз відповідних результатів. Тестування програмного забезпечення – це процес аналізу програмного засобу й відповідної документації, з метою виявлення дефектів і підвищення якості продукту [4].

Тестування програмного забезпечення (ПЗ) охоплює не тільки проведення тестів, але й інші елементи процесу забезпечення якості, такі як: планування та аналіз вимог; критерії початку тестування; стратегія тестування; проєктування тестових сценаріїв; виконання тест-кейсів, вимог; фіксація дефектів; аналіз результатів; написання звітів [5].

Автоматизоване тестування (АТ) – це набір технік, підходів і інструментальних елементів, які дозволяють вилучити тестувальника з виконання деяких завдань в процесі тестування. У автоматизованому тестуванні використовується три рівні: тестування на рівні коду; функціональне тестування; GUI – тестування (Graphical User Interface – графічний інтерфейс користувача) [5].

Однією із задач підвищення продуктивності роботи WEB-додатку є збільшення швидкості обробки даних шляхом розподілення завантаження сторінки таким чином, щоб користувач зміг взаємодіяти із нею ефективніше [2, 3]. WEB-сайти, які швидко завантажуються, підвищують загальний досвід взаємодії користувача з додатком. У WEB-реалізації найважливішим показником є час до взаємодії із WEB-сайтом. Найефективнішим способом оптимізації є послідовне завантаження елементів WEB-сайту, щоб користувач міг почати користуватися WEB-сайтом якомога швидше.

1.2 Аналіз відомих технічних рішень клієнтських модулів програмних систем автоматизованого тестування WEB-додатків

Нині велика кількість програмних систем створено як WEB-додатки (ті, що виконуються у WEB-браузері). Як і для будь-якого програмного забезпечення, ці програми повинні мати гарантію якості перед їх розгортанням кінцевим користувачам. У цьому сценарії тестування є важливою діяльністю, яка гарантує якість системи. У рамках своїх спроб зробити більше з меншими витратами, організації хочуть тестувати програмне забезпечення адекватно, але якомога швидше і ретельно. Для досягнення цієї мети організації звертаються до автоматизованого тестування [6].

Сучасна практика тестування WEB-додатків вимагає ручного створення тестових випадків, що є складним і виснажливим. Правильна інтеграція сторонніх служб у WEB-додатки є складною справою, і помилки можуть мати серйозні наслідки, коли сторонні служби використовуються для виконання важливих для безпеки завдань, таких як аутентифікація та авторизація [7].

Розробники часто неправильно розуміють вимоги до інтеграції та роблять критичні помилки під час інтеграції таких служб. Оскільки традиційні методи програмування важко застосувати до програм, що працюють на WEB-серверах із чорними скриньками, доцільно буде виявляти вразливості, досліджуючи поведінку системи.

Складно назвати JMeter [8], програмою легкою для розуміння недосвідченими користувачами. Головне вікно програми має досить багато різних налаштувань, які можуть фокусувати увагу на непотрібних деталях (рис. 1.1).

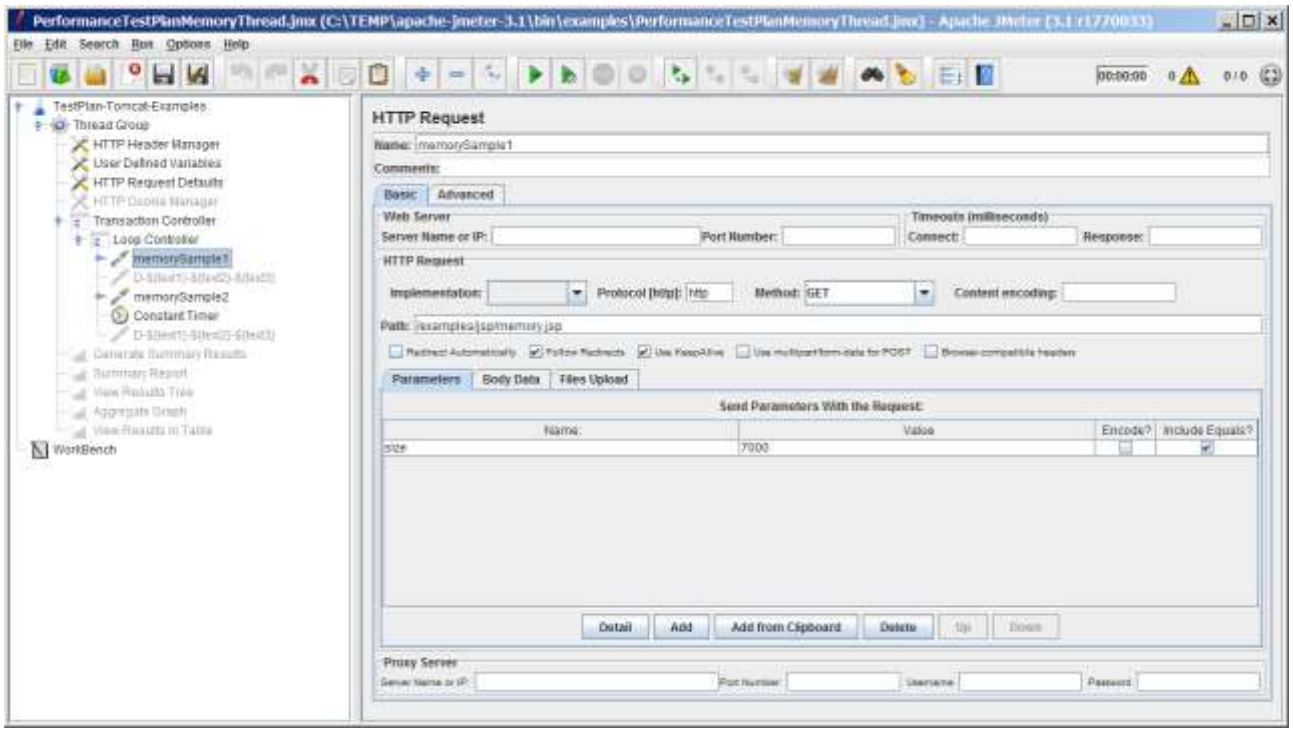


Рисунок 1.1 – Загальний вигляд інтерфейсного вікна програмного середовища автоматизованого тестування JMeter

Також JMeter тести досить складно коректно автоматизувати, так як JMeter спрямований на саме навантажуваче тестування - моделювання очікуваного використання шляхом імітації одночасного доступу кількох користувачів до WEB-служб, або стрес тестування – кожен WEB-сервер має максимальне навантаження, коли навантаження перевищує ліміт, WEB-сервер починає реагувати повільно та видає помилки, метою стрес-тестування є визначення максимального навантаження, яке може витримати WEB-сервер [8].

Результати тестування JMeter виводяться користувачу у вигляді окремих графіків або таблиць, посортованих відповідно до визначених запитів, які JMeter відправляє до WEB-додатку, на якого спрямовані тести. Через велику кількість таких запитів у одному тест-кейсі вихідні звіти буває досить складно зрозуміти без навчання. Приклад одного з таких звітів представлений на рисунку 1.2.

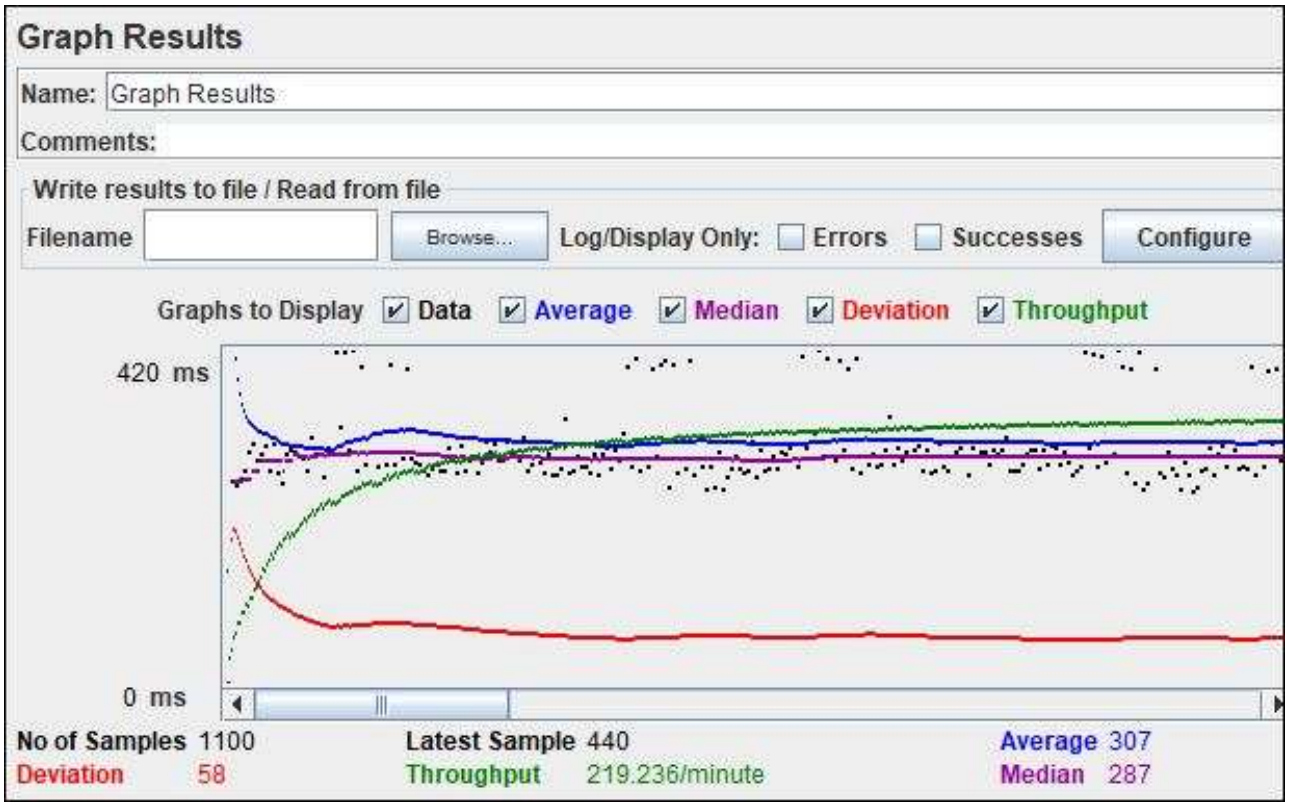


Рисунок 1.2 – Загальний вигляд інтерфейсного вікна вихідного звіт тест-кейсу представлений у вигляді графіку у програмного середовища автоматизованого тестування JMeter

Object-Driven Testing (ODT) – це процес тестування програмного забезпечення, який проводиться для тестування програмного забезпечення з використанням об’єктно-орієнтованих парадигм, таких як інкапсуляція, успадкування, поліморфізм тощо [9]. Програмне забезпечення зазвичай проходить багато рівнів тестування, від модульного до системного або приймального тестування. Як правило, тестування в модулі, невеликі «блоки» або модулі програмного забезпечення тестуються окремо з акцентом на тестуванні коду цього модуля. У тестуванні вищого порядку (наприклад, приймальному тестуванні) перевіряється вся система (або підсистема) з акцентом на тестуванні функціональності або зовнішньої поведінки системи. Цей метод тестування є не алгоритмічним, а орієнтованим на дані. Це техніка, яка базується

на ієрархії класів і чітко визначених об’єктів. Тут об’єкт визначається як сутність або екземпляр класу, який використовується для зберігання даних і надсилання та отримання будь-яких повідомлень, а клас можна визначити як групу об’єктів, які мають спільні властивості.

Часто створення сценаріїв є переважно завданням програмування. Якщо тест не потребує великої кількості даних і спрямований на перевірку лише одного об’єкта (наприклад, однієї форми заявки), то можна написати одну або дві підпрограми скриптів, які використовують кілька локальних змінних і, можливо, деякі глобальні константи. Однак більшість тестових проектів є великими, вони включають багато процедур сценаріїв і використовують багато тестових даних [10].

У цьому випадку програмування стає логічно простішим, якщо робити це об'єктно-орієнтованим способом, тобто якщо використовуєте власні об'єкти. Наприклад, можна створити окремий об’єкт для кожної перевіреної форми, а потім використовувати методи та властивості цього об’єкта для перевірки елементів керування у формі. Іноді потрібно створити не лише окремі об’єкти, але й складну ієрархію об’єктів, де властивості об’єктів містять спеціальні структури даних, масиви, посилання на інші об’єкти тощо. Щоб підтримувати можливості спеціальних об’єктів, TestComplete використовує ODT елемент проекту [10].

Елемент проекту ODT працює з настоюваними структурами даних і об’єктами за такими принципами:

– Дані організовані в групи. Кожна група містить набір змінних. Кожна змінна може зберігати звичайне значення, масив або об’єкт. Щоб налаштувати дані, використовуйте підпункт «Дані» елемента проекту ODT. Щоб створити та отримати доступ до користувацьких даних зі сценаріїв або тестів ключових слів, використовуйте методи та властивості об’єкта Data.

– Усі настроюванні об’єкти мають бути створені на основі класу, створеного за допомогою підпункту Classes елемента проекту ODT або за допомогою методів і властивостей об’єкта Classes. Тобто, перш ніж створити об'єкт, потрібно створити для нього клас.

Представлений вид автоматизованого тестування відрізняється своєю гнучкістю та може охоплювати різні варіації тестів, що потребує користувач. ODT – це потужний інструмент, що дозволяє створювати тестові скрипти високого класу, використовуючи динамічні класи та об’єкти, або зв’язувати методи класу зі створеними функціями, що зображено на рисунку 1.3 та результат роботи програми зображений на рисунку 1.4.

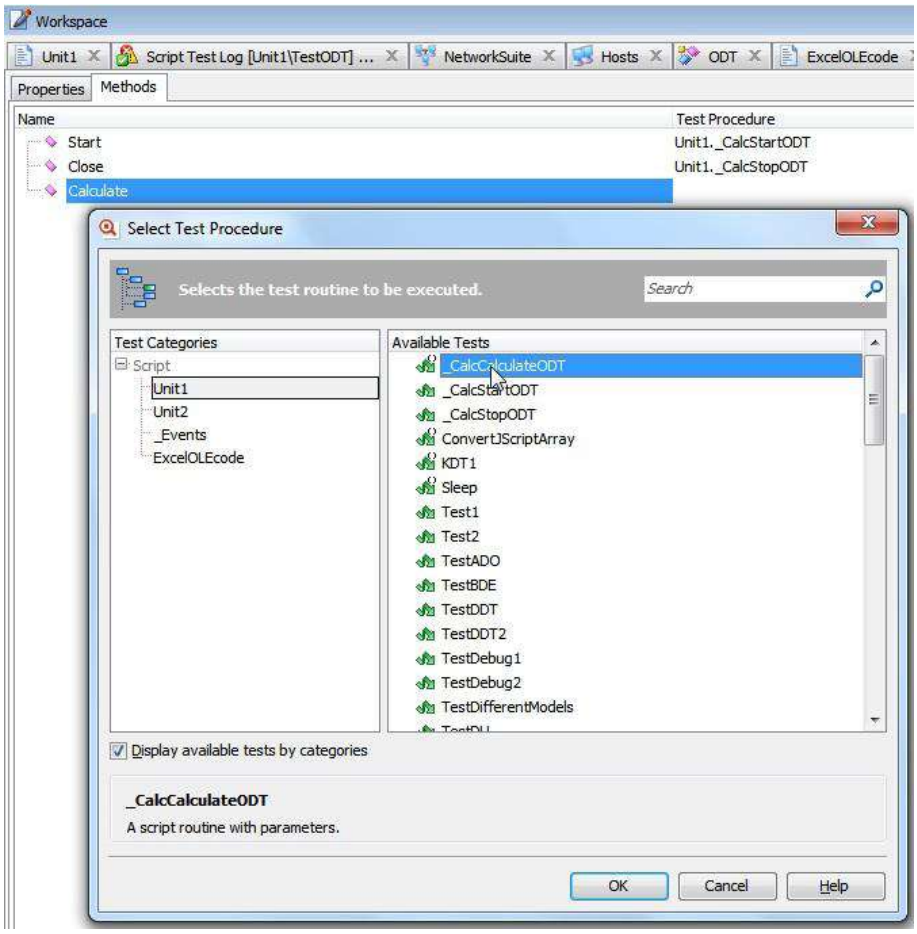


Рисунок 1.3 – Загальний вигляд інтерфейсного вікна зв’язування методів класу програмного середовища автоматизованого тестування ODT

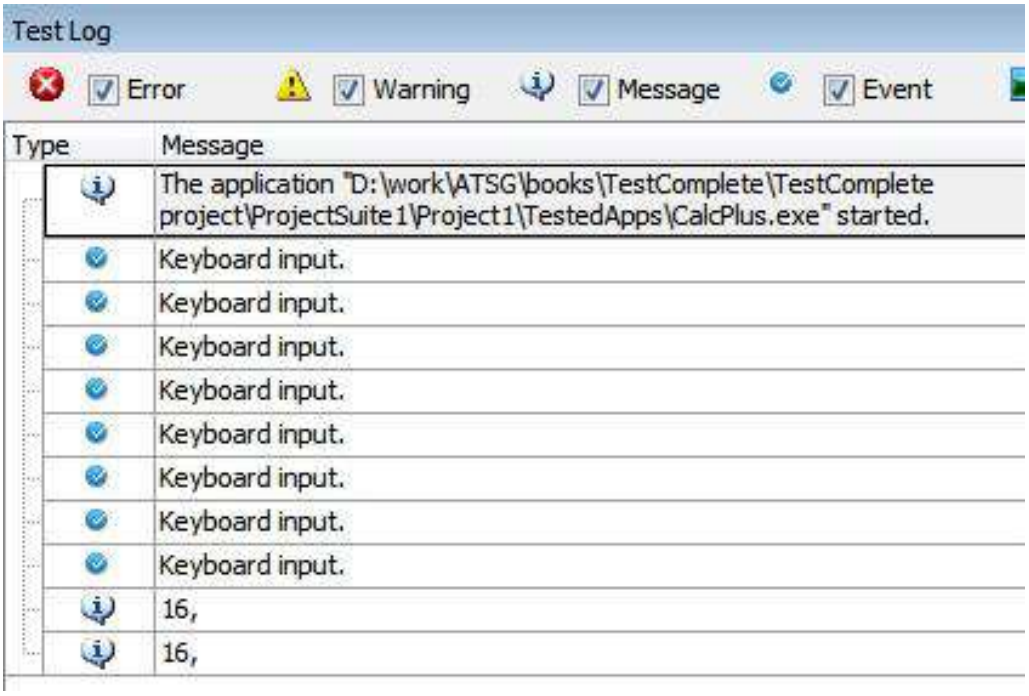


Рисунок 1.4 – Загальний вигляд інтерфейсного вікна вихідних даних після роботи скрипта у програмному середовищі автоматизованого тестування ODT

Відповідно до можливостей представлений інструментарій потребує необхідних навичок у написанні відповідних тест скриптів до поставленої задачі при автоматизації тестування. Саме тому пересічному користувачу буде досить складно протестувати WEB-додаток використовуючи ODT. У доповнення, функціональність ODT на даний момент є застарілою і майже не використовується у професійних середовищах, а її підтримка та оновлення були призупинені [11].

Саме тому, використання Object-Driven Testing, хоч і надає гнучкий та потужний інструментарій, але через складність написання тест-кейсів, призупинену підтримку та застарілу функціональність Object-Driven Testing використовувати не вдала ідея.

З інших, відомих технічних рішень систем автоматизованого тестування WEB-додатків – TestCafe. Це крос-платформний фреймворк для

функціонального тестування WEB-застосунків, випущений компанією DevExpress. TestCafe переважає там, де інші рішення для наскрізного тестування не вистачають. TestCafe не залежить від іншого програмного забезпечення для тестування, працює на платформі Node.js і використовує браузери, що наявні у користувача. А процес встановлення складається з однієї команди [11-12].

Розробники TestCafe виділяють наступні акценти при розробці даної системи [12]:

– основні компоненти API прості у використанні, навіть з незначними знаннями JavaScript;

– тести TestCafe можуть використовувати можливості сторонніх бібліотек і препроцесорів JavaScript;

– розширені функції TestCafe допомагають тестувати складні, чутливі до безпеки WEB-програми;

– підтримка емуляції HTTP-відповіді, щоб надсилати зразки даних у програму, усувати помилки підключення чи шахрайства;

– можливість виконати спеціальний код на стороні клієнта, щоб проаналізувати WEB-сторінку, перевірити її стан або навіть додати додаткові залежності;

– кілька вікон браузера, щоб перевірити складну взаємодію користувача. Легко та стабільно перемикається між iframes;

– виконання тестів у кількох браузерах одночасно, щоб швидко виявити специфічні для браузера помилки.

Також TestCafe було розроблено з урахуванням застосування автоматизації, з цією метою було імплементовано: автоматизоване очікування, скрипти автоматизованої автентифікації, «хуки» та інше [12, 13].

Взаємодія з користувачем, графічний інтерфейс і тестовий записувач, вбудовані в TestCafe, що робить його, більш доступним для людей без досвіду

програмування. Можливість увімкнути вбудований живий режим, щоб перезапускати тест кожного разу, коли змінюється файл тесту, допомога з налагодженням – призупиняє тести, щоб перевірити програму та усунути помилки [12].

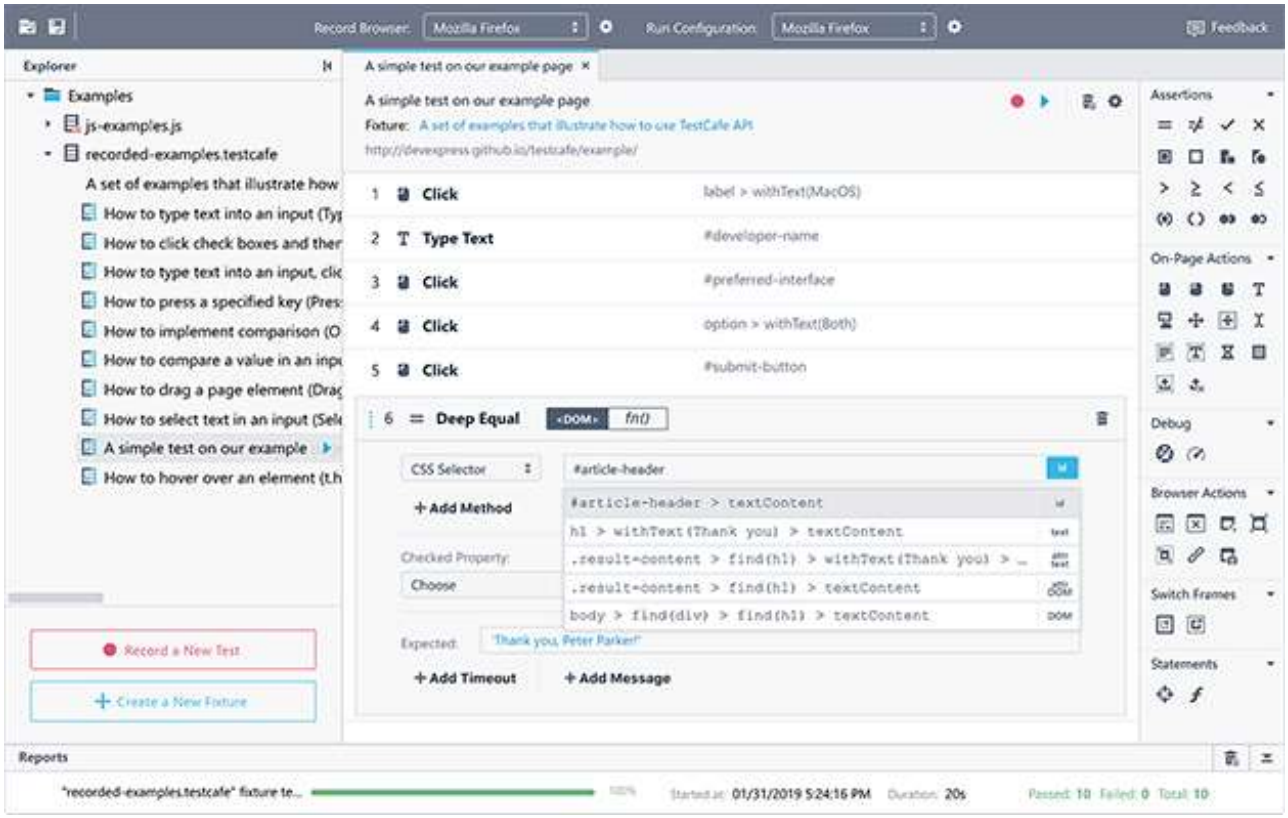


Рисунок 1.5 – Загальний вигляд інтерфейсного вікна програмного середовища автоматизованого тестування TestCafe

На рисунку 1.5 зображений вигляд вікна програмного середовища TestCafe. Проаналізувавши даний рисунок, можна зрозуміти, що програмне середовище TestCafe чимось схоже з середовищем автоматизованого тестування JMeter, але на відмінну від попереднього TestCafe має спрощений вигляд, що дозволяє менш досвідченим користувачам використовувати дане програмне середовище. Проте воно все ще містить забагато надлишкових налаштувань, що можуть змістити фокус з елементів, яких потрібує користувач. Більше за те, результати

проведених тестів, що зображені на рисунку 1.6, демонструються у вигляді таблиці, що може викликати труднощі, при аналізуванні недосвідченими користувачами, та на масштабних проектах досить складно виявити тест, який виявив помилку.

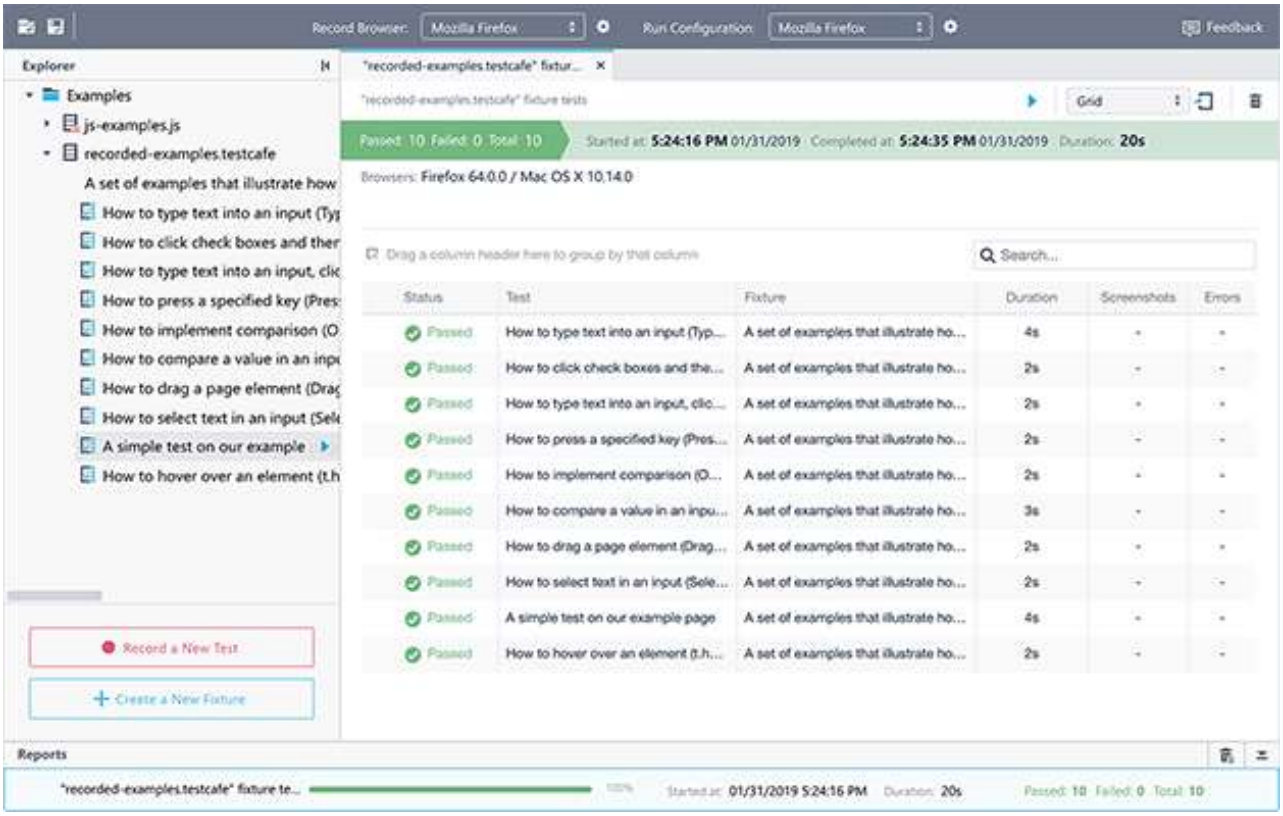


Рисунок 1.6 – Загальний вигляд інтерфейсного вікна результатів проведених тестів програмного середовища автоматизованого тестування TestCafe

1.3 Порівняльний аналіз характеристики клієнтських модулів програмних систем автоматизованого тестування WEB-додатків

Аналізуючи розглянуті, у розділі 1.2, приклади, клієнтських модулів програмних систем автоматизованого тестування WEB-додатків, можна легко помітити, що, здебільшого, вони усі схожі між собою, особливо JMeter та TestCafe. Найбільша різниця з ODT це те, що він не має власного IDE інтерфейсу,

а являє собою лише інструмент, який застосовується шляхом написання коду для імплементації тест кейсів, що, також, є найбільшим недоліком.

Ще один масштабний недолік ODT від JMeter та TestCafe – ODT на даний момент більше не підтримується розробниками, та є застарілим продуктом. А JMeter та TestCafe досі підтримуються, та регулярно виходять нові оновлення, що благодатно впливає на написані тест кейси, використовуючи ці системи тестування.

JMeter та TestCafe хоч і подібні, але й різниця між ними суттєва. А саме TestCafe має спрощений інтерфейс, що допомагає недосвідченим користувачам, але більшість функцій, що повинна тестувати TestCafe, користувач має описати самостійно використовуючи мову JavaScript. Та головним недоліком TestCafe є те, що більшість проблем, які можуть виникнути при написанні тест кейсів, вирішуються за допомогою платної підписки. На відміну від JMeter, усі його функції доступні одразу, та можуть бути використані у повному обсязі без додаткової платні.

Тому представлене порівняння можна описати у вигляді схеми набору окремих характеристик, що зображено у таблиці 1.1.

Таблиця 1.1 – Порівняння характеристик клієнтських модулів програмних систем автоматизованого тестування WEB-додатків

	JMeter	TestCafe	ODT
Час початкового відображення вмісту (мс)	800	700	1500
Час відображення найбільшої частини вмісту (мс)	1 500	1960	2320
Час до взаємодії зі сторінкою (мс)	980	630	4200
Загальний час блокування WEB-сторінки (мс)	80	220	570
Кумулятивний зсув макета	0,015	0,168	0,007
Час завантаження WEB-сторінки (мс)	900	800	1700

Продовження табл. 1.1

Крос-платформеність та крос-браузерність	+	+	-
Кількість TPS (ticks per second)	20	20	15
Активна підтримка та покращення	+	+	-
Графічний інтерфейс	+	+/-	-
Підтримка стрес тестування	+	+/-	-
Повна функціональність без оплати	+	-	+
Можливість виведення звітів тестування у різних формах	+	-	-
Стабільність виконання тестів	+/-	+	-
Відсутність необхідності в додатковому програмному забезпеченні.	-	+	+
Використання сторонніх модулів	-	+	+
Спрощена автоматизація	-	+	+/-

Підсумовуючи вищесказане можна виділити наступні переваги та недоліки середовища автоматизованого тестування JMeter.

Переваги JMeter:

- простий у використанні без глибоких знань програмування. Можливість використовувати CLI;
- стандартизований час початкового відображення вмісту – 800 мс;
- забезпечує інтеграцію з Jenkins і звітування;
- легка установка на будь-яку операційну систему;
- найменший час відображення найбільшої частини вмісту, відносно інших розглянутих програмних реалізацій – 1500 мс;
- такі ключові функції, як Thread Group, допомагають визначити, достатній рівень продуктивності програмного забезпечення;
- тестове IDE дозволяє записувати тести з браузерів або рідних програм;

- один з еталонних значень загального часу блокування WEB-сторінки – 80 мс;
- дозволяє з легкістю тестувати API, тестувати бази даних і тестувати MQ;
- коли є велика кількість TPS, можна отримати більше транзакцій за секунду, враховуючи гіперобмеження.

Недоліки JMeter:

- автоматизація складна з JMeter;
- вихідні звіти JMeter важко зрозуміти без навчання;
- він не підтримує запити JavaScript і AJAX;
- складні програми, які використовують динамічний вміст або використовують JS для зміни запитів, може бути важко перевірити за допомогою JMeter;
- важко отримати дані з одного місця або виконати налаштування.

Наступні переваги та недоліки системи TestCafe, а саме переваги:

– активно обслуговується та внутрішня підтримка;
– системні API високого рівня та сценарії на стороні клієнта;
– достатньо низький час початкового відображення вмісту – 700 мс;
– тести, які виконуються паралельно;
– одна хвилина для налаштування;
– найкращий можливий час до взаємодії зі сторінкою – 630 мс;
– не вимагає WebDriver або іншого програмного забезпечення.

Недоліки:

– для вирішення серйозних проблем потрібно купувати повну версію;
– необхідності встановлення браузерних плагінів;
– відсутність детермінованості тестового сценарію;
– асинхронність – невелика візуальна затримка для машини це дія з побічними ефектами;

- завелике значення загального часу блокування WEB-сторінки – 220 мс;
- проблеми з авторизацією;
- високе значення кумулятивного зсуву макета – 0,168;
- системні API високого рівня можуть викликати проблеми для людей які з ними не працювали раніше;
- нестабільні функціональні тести.

1.4 Висновок до розділу 1

У даному розділі магістерської кваліфікаційної роботи було розкрито поняття тестування, предметну область дослідження, визначено принципи тестування WEB-додатків. Досліджено наявні клієнтські модулі програмних систем автоматизованого тестування WEB-додатків.

Було виконано порівняльний аналіз розглянутих програмних систем автоматизованого тестування WEB-додатків. Виділено результати порівняльного аналізу у вигляді схеми набору окремих характеристик.

2 ПРОЕКТУВАННЯ ІНТЕЛЕКТУАЛЬНОЇ МОДЕЛІ СИСТЕМИ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ WEB-ДОДАТКІВ

2.1 Розробка підходів проведення автоматизованого тестування WEB-додатків

Тестування WEB-додатків – це метод перевірки того, що фактичний програмний продукт відповідає очікуваним вимогам, і гарантії, що продукт вільний від дефектів. Він передбачає виконання програмного забезпечення та системних компонентів за допомогою ручних або автоматизованих інструментів для оцінки продуктивності [14]. На рисунку 2.1 продемонстровані рівні тестування.

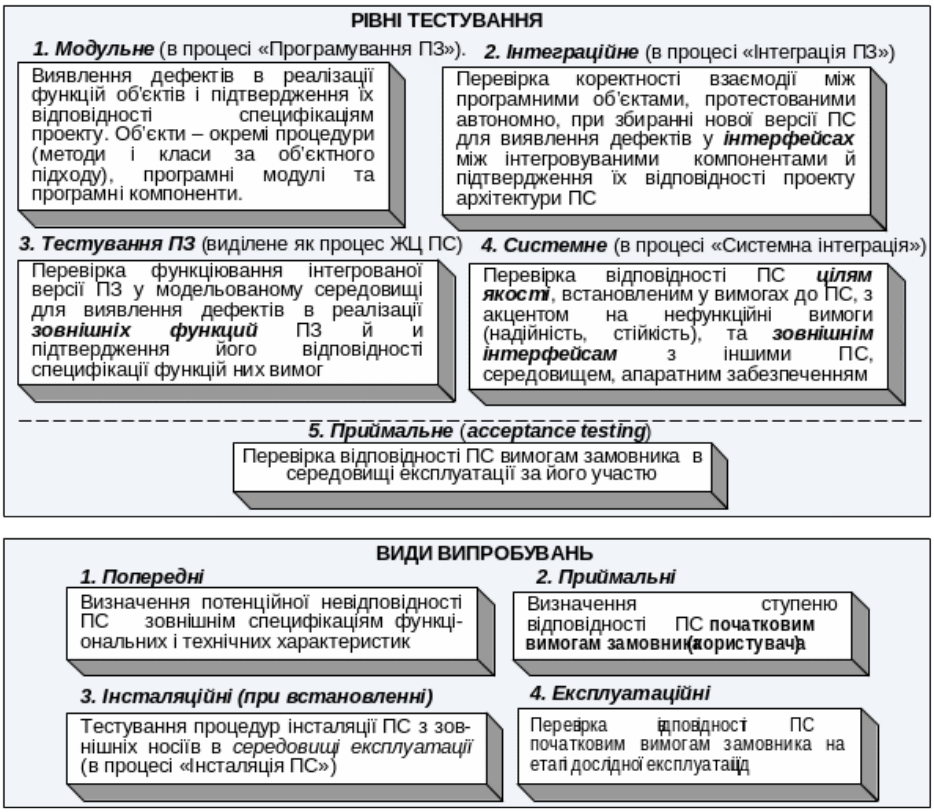


Рисунок 2.1 – Рівні тестування програмного забезпечення

Існує п’ять рівнів тестування, які можуть допомогти перевірити поведінку та ефективність тестування програмного забезпечення, та чотири рівні випробувань [14].

Статичне тестування – це тестування програмного забезпечення без запуску коду. Цей термін відноситься до процесу виявлення та усунення помилок і дефектів у супровідних документах, таких як специфікації вимог до програмного забезпечення. Зазвичай це виконується перед виконанням більш поглибленого тестування [15].

Можна поділити статичне тестування на два типи: «Огляд» (Review) та «Статичний аналіз» (Static Analysis).

«Огляд» відноситься до аналізу та вирішення будь-яких невідповідності або проблем із документами. Його можна використовувати для визначення вимог, проектів, тестів тощо.

У свою чергу «Огляд» поділяється на [15]:

- Неформальний. При неофіційному розгляді творець документів показує вміст документів аудиторії. Кожен присутній висловлює свою думку, що дозволяє виявити недоліки на ранній стадії.
- Наскрізний перегляд (Walkthroughs). Виконуються досвідченою людиною або експертом для перевірки відсутності дефектів, з метою попередження виникнення проблем на етапі розробки або тестування.
- Експертна оцінка. Означає перевірку документів для виявлення і виправлення дефектів. В основному це виконується у команді.
- Інспектування ПЗ. Це, в більшості випадків, перевірка документу вищим органом, наприклад, перевірка вимог до програмного забезпечення.

Код, створений розробниками, перевіряється на якість за допомогою статичного аналізу. Різні методики допомагають визначити, чи відповідає програмний код стандартам чи ні. Одним із прикладів є статичний аналіз, який

виявляє такі помилки, як неправильна логіка. Це включає виявлення дефектів у кодовій базі, як [15, 16]: змінні, які не використовуються; мертвий код; нескінченні цикли; змінні з невизначеними значеннями; неправильний синтаксис.

Потік даних пов’язаний зі статичним аналізом обробки потоку програми. Додатковий аналіз можна виконати, вимірявши кількість шляхів у програмі за допомогою цикломатичної складності. Обидва ці методи зазвичай виконуються вручну, але для статичного аналізу можна використовувати спеціальні інструменти. Перегляд коду зазвичай виконує розробник, який створив програмний продукт, що перевіряється. Ці огляди зазвичай призначені для виявлення потенційних проблем із програмним забезпеченням на ранніх стадіях розробки [16].

Динамічне тестування – це метод тестування програмного забезпечення, який використовується для перевірки динамічної поведінки програмного коду. Основною метою динамічного тестування є використання динамічних або непостійних змінних для перевірки поведінки програмного забезпечення та пошуку слабких місць у середовищі виконання програмного забезпечення. Щоб перевірити динамічну поведінку, потрібне виконання коду [17].

У загальному архітектуру динамічного тестування можна представити у вигляді схеми залежностей видів систем та відповідних тестів, що представлено на рисунку 2.2.

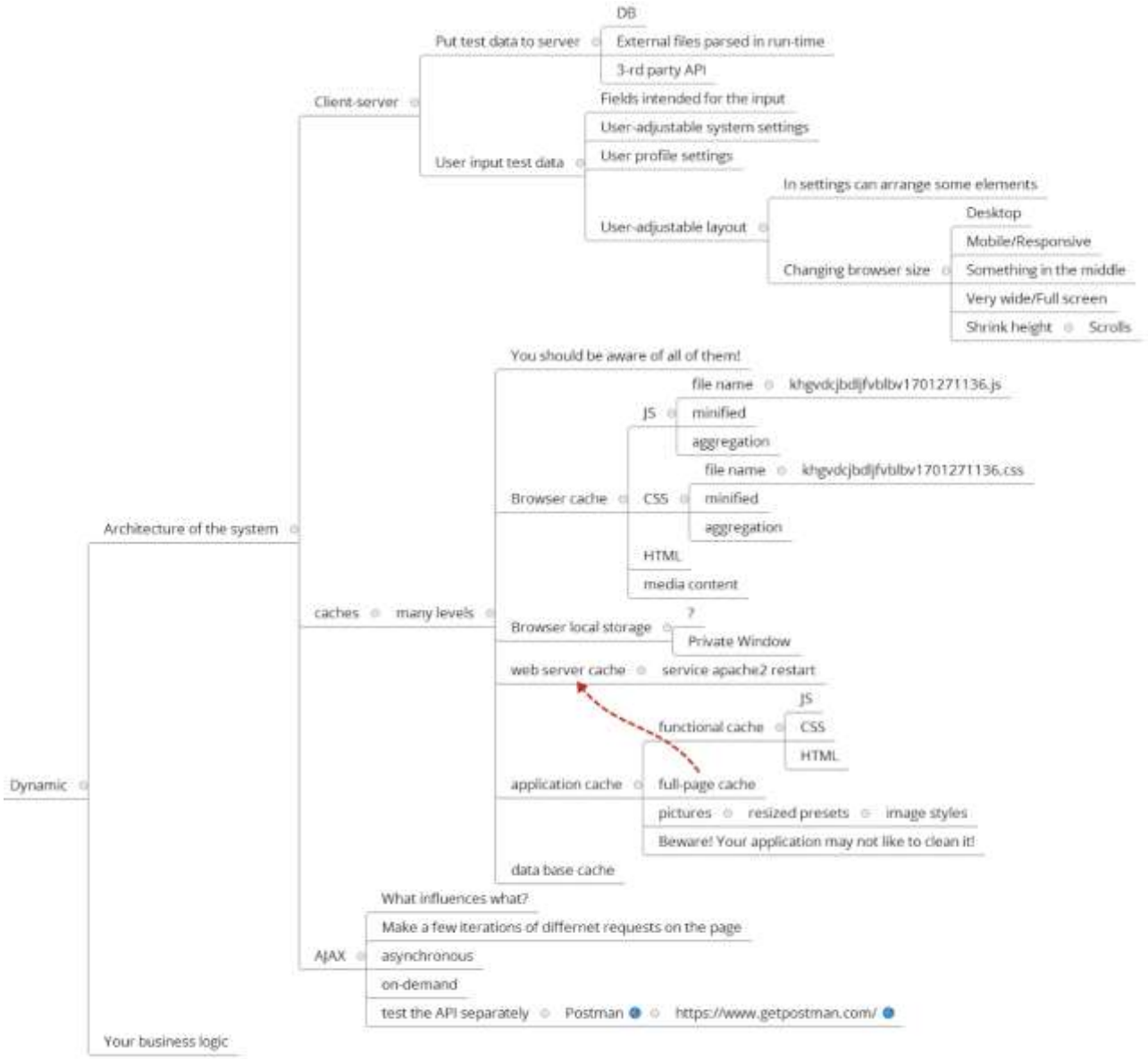


Рисунок 2.2 – Схема архітектури динамічного тестування

Динамічне тестування включає в себе тестування ПЗ в режимі реального часу шляхом надання вхідних даних і вивчення результату поведінки програми.

Порівняльні характеристики статичного та динамічного тестування представлені у таблиці 2.1.

Таблиця 2.1 – Порівняльний аналіз динамічного та статичного тестування

Динамічне тестування	Статичне тестування
Етап валідації ПЗ	Етап верифікації ПЗ
Включає в себе виконання програмного коду	Не потребує виконання програмного коду
Забезпечує функціональність продукту	Орієнтоване на запобігання дефектів
Виконується на більш пізніх етапах розробки програмного забезпечення	Виконується на ранніх етапах розробки програмного забезпечення
Велика вартість виправлення дефектів	Менша вартість виправлення дефектів
Покриває обмежену область коду, потребує меншого охоплення	Забезпечує більш ширше охоплення за коротший проміжок часу
Включає в себе як функціональне так і нефункціональне тестування	Включає в себе різні методи оцінки, наскрізний перегляд і багато іншого
Мета – пошук і усунення дефектів програмного забезпечення	Мета – запобігання дефектів програмного забезпечення
Виявляє менше дефектів	Комплексне тестування коду, яке допомагає знайти більше дефектів у системі
Виконується після завантаження коду	Виконується перед завантаженням коду

Розуміння того, наскільки критичним є статичне тестування для загальної розробки програмного забезпечення, дозволяє легко прийняти рішення чи завершити проект без будь-яких дефектів. Багато людей вважають статичне тестування тривалим процесом без позитивних результатів. Однак це помилкове

рішення, засноване на наданій інформації. Жоден готовий продукт не повинен мати проблем під час розробки. Таким чином, тестування на останніх етапах розробки варте часу та зусиль. Динамічне тестування має важливе значення для забезпечення якості, ефективної функціональності та надійності програмного забезпечення. Він передбачає виконання тестів програмного забезпечення, таких як продуктивність, надійність та інші важливі аспекти. Виконуючи ці тести, команда може безпосередньо підтвердити якість і ефективність програмного забезпечення [17].

2.2 Математична модель динамічного автоматизованого тестування WEB-додатків

Подібно до інших форм генерації динамічних тестів, DART (Directed Automated Random Testing) – спрямоване автоматизоване випадкове тестування, складається з запуску тестованої програми «*P*» як конкретно, виконання фактичної програми, так і символічно, обчислення обмежень на значення, що зберігаються в програмних змінних і виражені через вхідні параметри. Дане взаємне виконання вимагає, щоб програма «*P*» була оснащена інструментами на рівні RAM (Random Access Memory). Пам’ять (*M*) – це відображення адрес пам’яті на, наприклад, 32-розрядні слова. Позначення + для відображень означає оновлення [21].

У загальному проведення динамічного тестування можна описати окремими кроками, зв’язок яких схематично зображений на рисунку 2.3.

Рисунок 2.3 – Схема покрокового проведення динамічного тестування

Наприклад:

$$M' := M + [m \rightarrow v], \tag{2.1}$$

де m – це адреса пам’яті, переривання, що відповідає програмній помилці, або зупинка, що відповідає нормальному завершенню [21], а v – розрядність слова, має таке ж відображення що і M, за виключенням

$$M'(m) = v\,. \tag{2.2}$$

Ідентифікуючи символічні змінні за їхніми адресами. Таким чином, у виразі m позначає або адресу пам’яті, або символічну змінну, ідентифіковану адресою m, залежно від контексту [21].

Програма «*P*» маніпулює пам'яттю за допомогою операторів, які є спеціально розробленими абстракціями фактично виконаних машинних інструкцій. Оператор може бути умовним оператором у формі:

$$if\ (e), \text{then goto}\ ,$$

де e – вираз над символьними змінними і є міткою оператора, оператором присвоювання.

У вигляді:

$$m \leftarrow e. \tag{2.3}$$

Конкретна семантика машинних інструкцій RAM для «*P*» відображається в конкретній визначеній кількості (*e*, *M*), який обчислює вираз *e* у контексті *M* і повертає 32-бітне значення для *e*. Програма «*P*» визначає послідовність вхідних адрес M_0, адреси вхідних параметрів «*P*». Вхідний вектор *I* зв'язує значення з кожним вхідним параметром і визначає початкове значення M_0 і *M* [22].

Нехай *C* – набір умовних операторів та *A* набір операторів присвоєння в «*P*». Виконання програми *w* є скінченною послідовністю:

$$Execs := (A \cup C) * (abort \mid halt). \tag{2.4}$$

Конкретна семантика «*P*» на рівні машини RAM дозволяє нам визначити для кожного вхідного вектора *I* послідовність виконання: результат виконання «*P*» на *I*. Нехай *Execs*(*P*) буде набором таких виконань, створених усіма можливими *I*. Розглядаючи кожен оператор як вузол, *Execs*(*P*) формує дерево, яке називається деревом виконання. Його вузли призначення мають одного

наступника, умовні вузли якого мають одного або двох наступників. І його кінцеві нащадки позначено як «abort» – перервати, або «halt» – зупинити. Метою DART є дослідження всіх шляхів у дереві виконання *Execs*(*P*) [23].

DART підтримує символічну пам'ять *S*, яка відображає адреси пам'яті на вирази. Спочатку *S* є відображенням, яке відображає кожен $m \in M_0$ на себе. Коли вираз виходить за межі теорії *T*, DART просто повертається до конкретного значення виразу, яке використовується як результат. У такому випадку ми також встановлюємо прапорець «Виконано» на 0, який ми використовуємо для відстеження повноти. За допомогою цієї стратегії оцінювання символічні змінні виразів у *S* завжди містяться в M_0 [23].

Щоб здійснити систематичний пошук у дереві виконання, програма запускається багаторазово. Кожен запуск (крім першого) виконується за допомогою запису умовних операторів, виконаних напередодні. Для кожної умови ми записуємо значення «Виконано», яке дорівнює 0, якщо в попередніх запусках виконувалася лише одна гілка умови (з тією самою історією до точки розгалуження), і в інших випадках. Ця інформація, пов’язана з кожним умовним оператором останнього шляху виконання, зберігається у змінній списку під назвою стек, яка зберігається між виконаннями (рис. 2.4). Для *i*, ($0 \le i < |stack|$), таким чином, *stack*[*i*] є записом, що відповідає *i*+1-й виконаній умові [22, 23].

Точніше, тестовий драйвер DART, поєднує випадкове тестування (цикл повторення) із спрямованим пошуком (цикл «while»). Якщо інструментована програма викидає виняток, це означає, що виявлено помилку. Прапор «Виконано» зберігається, якщо не сталася «погана» ситуація, яка може призвести до незавершеності. Таким чином, якщо спрямований пошук припиняється, тобто якщо спрямованість внутрішнього циклу більше не виконується, зовнішній цикл також припиняється, якщо прапор все ще зберігається.

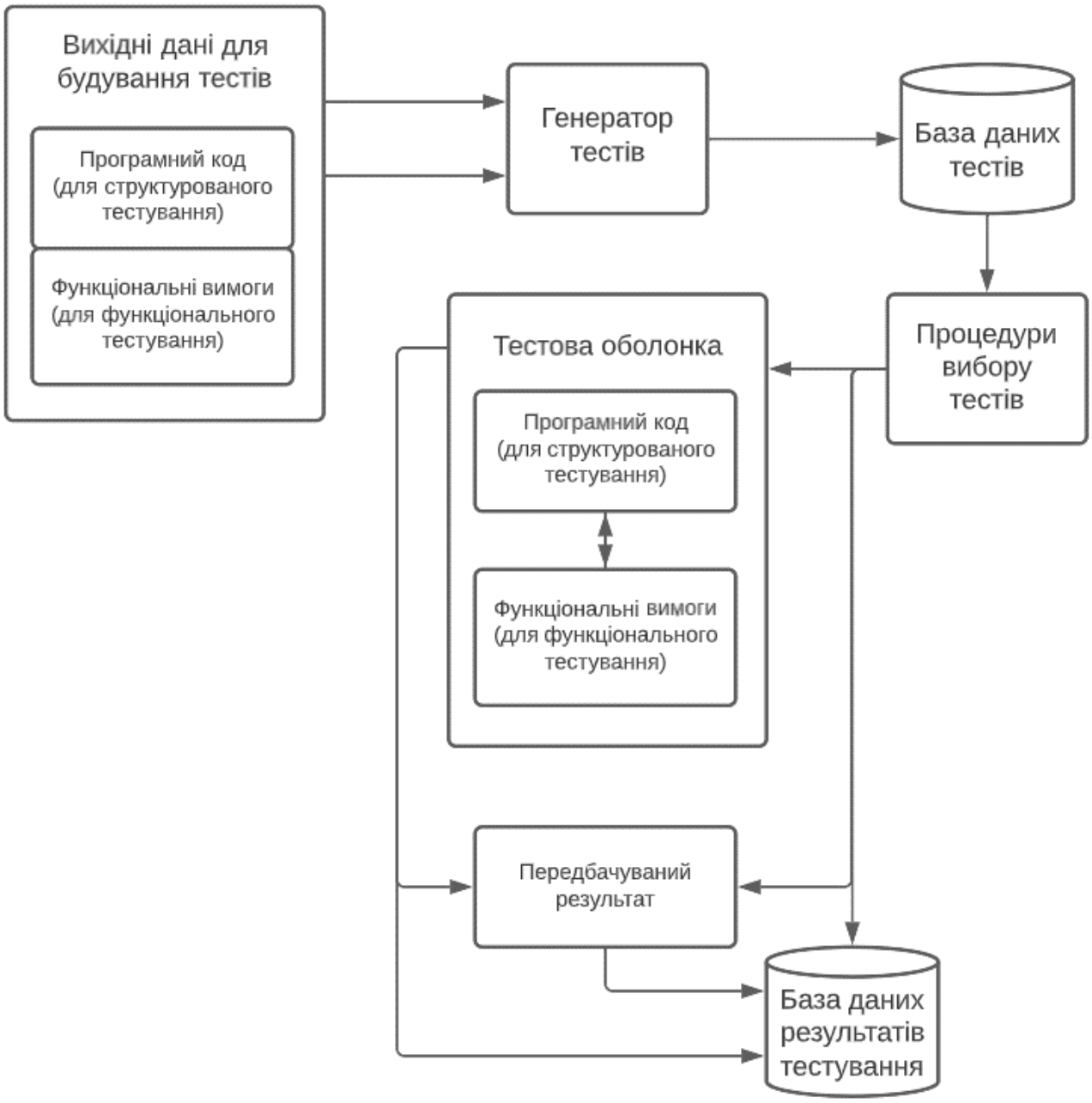


Рисунок 2.4 – Схема проведення динамічних тестів

У цьому випадку DART завершує роботу та безпечно повідомляє, що всі можливі шляхи програми досліджено. Але якщо прапор був вимкнений в якийсь момент, зовнішній цикл продовжується вічно [23].

Розглянемо просту лінійну регресійну модель динамічного коригування з серійно корельованими помилками:

$$y_t = \gamma y_{t-1} + X_t'\beta + u_t, \qquad t = 1,2,\dots,T, \tag{2.5}$$

$$u_t = \rho u_{t-1} + \varepsilon_t, \qquad |\rho| < 1, \tag{2.6}$$

де y_t є залежною змінною, X_t є вектором (Kx1) незалежних змінних, β є вектором (Kx1) параметрів, $|\gamma| < 1$ та y_0 є деяким фіксованим і відомим числом, $\{\varepsilon_t\}$ є незалежна та однаково розподілена випадкова величина з нульовим середнім відхиленням і постійною дисперсією σ^2_ε [24].

Припускаючи, що значення ρ відоме. Модель (2.5) і (2.6) можна привести до стандартної форми за допомогою

$$(y_t - \rho y) = \gamma(y_{t-1} - \rho y_{t-2}) + (X_t - \rho X_{t-1})'\beta + \varepsilon_t, \tag{2.7}$$

або

$$\tilde{y_t} = \gamma \tilde{y}_{t-1} + \tilde{X}_t'\beta + \varepsilon_t, \tag{2.8}$$

за нульовою гіпотезою. Слід зауважити, що трансформована модель (2.7) зводиться до стандартної моделі динамічної лінійної регресії. Таким чином, можна діяти як зазвичай, і рекурсивно оцінити вектор коефіцієнтів δ=(γ, β), та потім обчислити рекурсивні залишкові кількості:

$$w_r = (\tilde{y}_r - \tilde{z}_r'\hat{\delta}^{r-1})/f_r, \quad K+2 \le r \le T, \tag{2.9}$$

де

$$\tilde{z}_r = \left(\tilde{y}_{r-1}, \tilde{X}'_r\right), \tag{2.10}$$

$$f_r = \left[1 + \tilde{z}'_r \left(\tilde{Z}'^{r-1} \tilde{Z}^{r-1}\right) \tilde{z}_r\right]^{\frac{1}{2}}, \tag{2.11}$$

$$\tilde{Z}^{r-1} = (z_1, z_2, \ldots, z_{r-1})', \tag{2.12}$$

та $\hat{\delta}^{r-1}$ – це звичайна оцінка методом найменших квадратів для δ з перших (r–1) спостережень [24, 25].

Тоді динамічний тест буде приймати наступний вигляд:

$$S_T = \max_{K+2 \le r \le T} \left\{ \left| \frac{W^{(r)}}{\sqrt{T-K-1}} \right| \left(1 + 2\frac{(r-K-1)}{(T-K-1)}\right)^{-1} \right\}. \tag{2.13}$$

де $W^{(r)} = \frac{1}{\hat{\sigma}} \sum_{t=K+2}^{r} w_t$, $\hat{\sigma} = \left(\frac{1}{T-K-2} \sum_{t=K+2}^{T} (w_t - \overline{w})^2\right)$.

2.3 Розробка методу автоматизованого динамічного тестування WEB-додатків

DART – є варіантом проведення динамічних тестів, який поєднує тестування з методами перевірки моделі для систематичного виконання всіх можливих програмних шляхів під час використання інструментів перевірки під час виконання (таких як «Purify», для виявлення різних типів помилок) . У DART кожен новий вхідний вектор намагається змусити виконання програми через якийсь новий шлях. Повторюючи цей процес, цей спрямований пошук намагається змусити програму пройти всі можливі шляхи виконання у спосіб, подібний до тестування системи та перевірки динамічної моделі програмного

забезпечення. На практиці DART зазвичай забезпечує краще покриття, ніж чисте випадкове тестування [18].

Ключовий момент DART полягає в тому, що неточність у символьному виконанні можна зменшити за допомогою конкретних значень і рандомізації: щоразу, коли символічне виконання не знає, як створити обмеження для оператора програми в залежності від деяких вхідних даних, можна завжди спростити це обмеження за допомогою конкретні значення цих вхідних даних [19].

Цей момент можна зобразити на прикладі. Незважаючи на те, що статично неможливо згенерувати два значення для вхідних даних x і y, щоб обмеження $x==hash(y)$ задовольнялося (або порушувалося), легко згенерувати для фіксованого значення у значення x що дорівнює $hash(y)$, оскільки останній відомий під час виконання. Вибираючи випадковим чином, а потім фіксуючи значення у, ми можемо під час наступного запуску встановити значення іншого вхідного параметра x як $hash(y)$ або щось інше, щоб примусово виконати гілки тесту у функції [20].

Таким чином, проведення динамічного тесту можна розглядати як розширення статичного тесту з додатковою інформацією про час виконання, і тому вона є більш загальною та потужною (рис. 2.5). Також, даний метод може використовувати механізм символічного виконання та використовувати конкретні значення для спрощення обмежень за межами області дії засобу вирішення обмежень. Ось чому вважається, що проведення динамічних тестів має можливість запропонувати ефективні практичні інструменти для створення тестів, застосовні до реального програмного забезпечення [20].

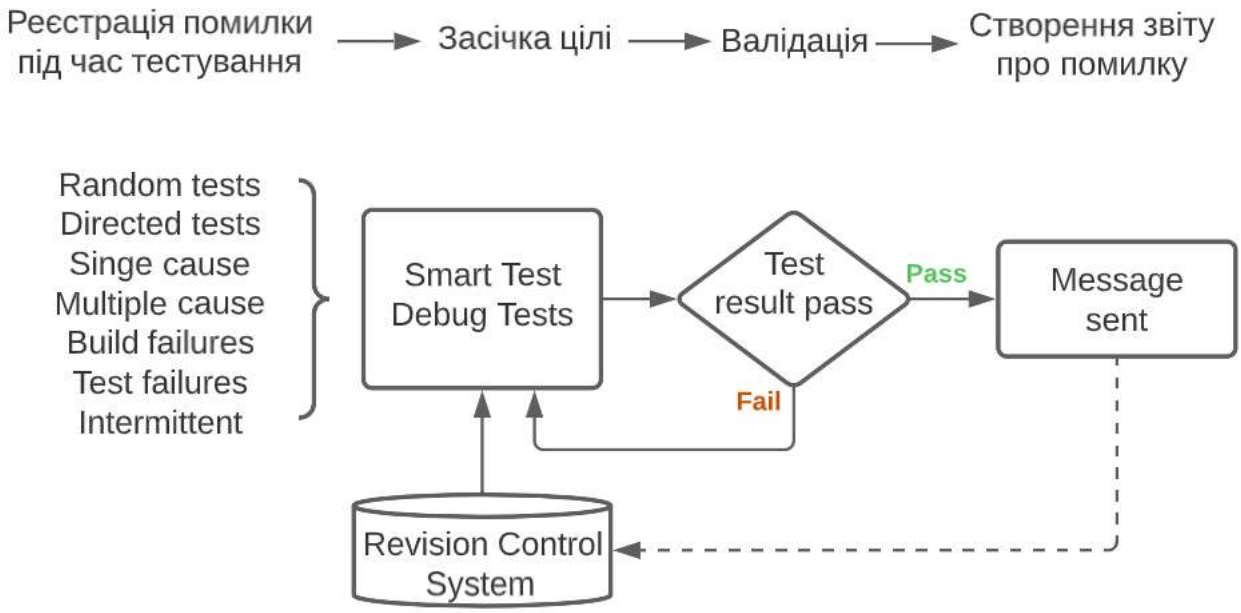


Рисунок 2.5 – Схема проведення динамічного тесту

Очевидно, що систематичне виконання всіх можливих програмних шляхів не масштабується до великих реалістичних програм. Тому у цьому методі розглядається основне обмеження та пропонується виконувати проведення динамічного тесту композиційно, адаптуючи відомі методи для міжпроцедурного статичного аналізу, які використовувалися, щоб зробити статичний аналіз масштабованим до дуже великих програм. Зокрема, алгоритм, який отримав назву SMART (Systematic Modular Automated Random Testing), який розширює DART шляхом ізольованого тестування функцій, кодування результатів тестування, як підсумків функцій, виражених за допомогою вхідних передумов і вихідних післяумов, а потім повторного використання цих підсумків під час тестування вище-рівневих функцій. Для фіксованої можливості міркування композиційний підхід до створення динамічних тестів є надійним і повним порівняно з монолітним генеруванням динамічних тестів [19, 20].

SMART – це альтернативний алгоритм пошуку, який не ставить під загрозу повноту пошуку, але набагато ефективніший, ніж алгоритм пошуку DART. Загальна ідея цього алгоритму полягає в композиційному виконанні динамічного

тесту шляхом адаптації (дуалізації) відомих методів міжпроцедурного статичного аналізу до контексту автоматизованого створення динамічного тесту. Зокрема, SMART для систематичного модульного автоматизованого випадкового тестування, який тестує функції окремо, збирає результати тестування як підсумки функцій, виражені за допомогою попередніх умов на входах функції та післяумов на виходах функцій, а потім повторно використовує ці підсумки під час тестування функції вищого рівня (рис. 2.6) [18, 20].

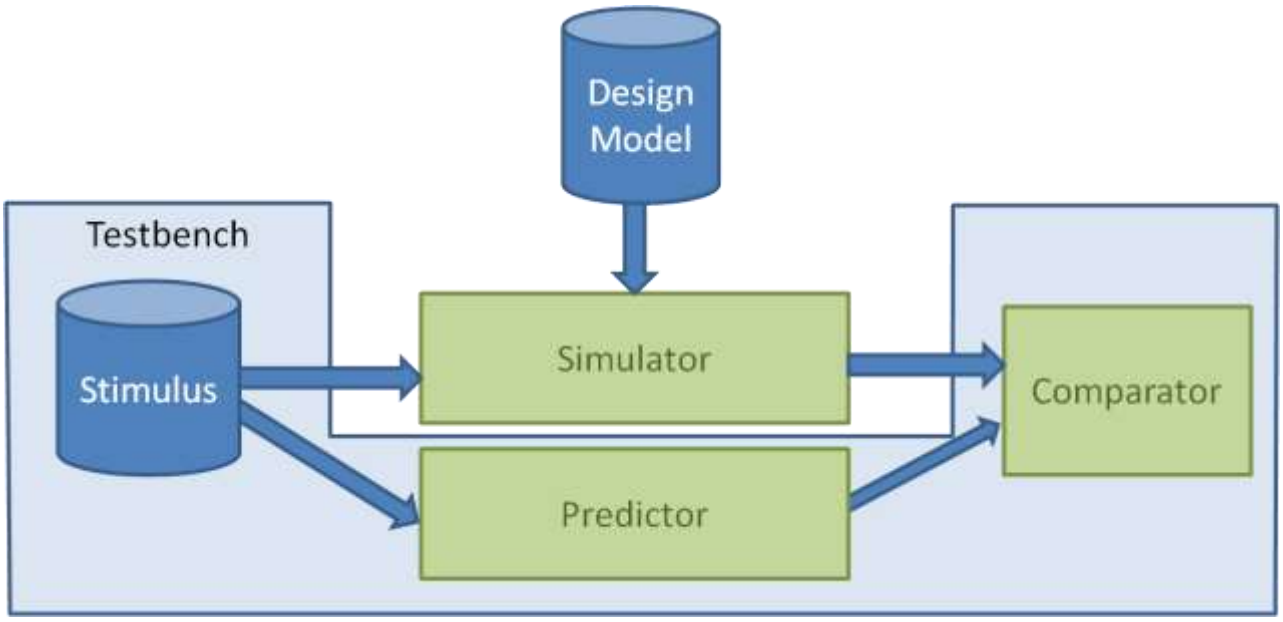


Рисунок 2.6 – Схематичний відображення дуалізації методів при виконанні динамічних тестів

Припустимо, що програма «P», яка складається з набору функцій. Якщо функція f є частиною «*P*», то $f \in P$. Далі використовуючи загальний термін функції для позначення будь-якої частини програми «*P*», яку потрібно проаналізувати ізольовано, а потім провести узагальнення її поведінки. Щоб спростити представлення, припускаємо, що функції в «*P*» не виконують рекурсивних викликів, тобто, що граф потоку викликів «*P*» є ациклічним. Також припускаємо, що всі виконання «*P*» завершуються. Потрібно наголосити, що

обидва припущення не заважають «P» мати нескінченну кількість шляхів виконання, як у випадку, якщо «P» містить цикл, кількість ітерацій якого може залежати від деякого необмеженого вхідного сигналу [20].

Іншими словами, виконується композиційне виконання динамічного тесту без будь-якого зменшення покриття програмного шляху. Враховуючи обмеження щодо максимальної кількості можливих шляхів в окремих функціях програми, кількість виконань програми є лінійною в цій межі, тоді як кількість виконань програми, дослідженої DART, може бути експоненціальною [20].

2.4 Висновок до розділу 2

У даному розділі магістерської кваліфікаційної роботи було розкрито підходи статичної та динамічної моделей тестування WEB-додатків. Проведено порівняльний аналіз зазначених підходів. Виділено результати порівняльного аналізу у вигляді схеми набору окремих характеристик. Розроблено математичну модель динамічного тестування. Розроблено метод автоматизованого тестування WEB-додатків відповідно до розглянутої математичної моделі динамічного тестування.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ КЛІЄНТСЬКОЇ ЧАСТИНИ СИСТЕМИ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ WEB-ДОДАТКІВ

3.1 Вибір інструментарію при розробці клієнтської частини системи автоматизованого тестування WEB-додатків

Інструменти WEB-розробки допомагають розробникам працювати з різними технологіями. Інструменти WEB-розробки повинні мати можливість забезпечити більш швидку розробку за менших витрат. Вони значно відрізняються від інтегрованих середовищ розробки тим, що не допомагають у фактичному створенні WEB-сайту, а лише надають інструменти для перегляду та тестування користувацького інтерфейсу цього WEB-сайту або WEB-програми. Це означає, що хоча WEB-дизайнери можуть зосередитись на дизайні візуальної частини сайту, WEB-розробники повинні думати про те, як технічна сторона буде з ним взаємодіяти. Інструменти WEB-розробки включають такі речі, як WEB-сервери, WEB-сторінки, онлайн-документацію, клієнтські фреймворки, такі як Java та HTML, мови сценаріїв, такі як JavaScript та ASP, засоби розробки баз даних та системи управління вмістом. WEB-дизайнери також можуть працювати з інструментами візуалізації даних, які допомагають їм візуалізувати інформацію на сайті або на сторінці WEB-сайту [21].

HTML (Hypertext Markup Language) – це стандартизована мова розмітки документів для перегляду WEB-сторінок у браузері. WEB-браузер використовує протокол HTTP/HTTPS, щоб отримати HTML-документ із сервера або відкрити його з локального диска, а потім інтерпретувати код як інтерфейс, який відображатиметься на екрані монітора [22].

Елементи HTML є будівельними блоками сторінок HTML. Використовуючи структуру HTML, зображення та інші об’єкти, наприклад інтерактивні форми, можна вставляти у відтворену сторінку. HTML надає можливість створювати структуровані документи шляхом позначення структурної семантики тексту, наприклад заголовків, абзаців, списків, посилань, цитат та інших елементів. Елементи HTML розділені тегами, записаними кутовими дужками. Такі теги, як <img /> або <input />, відображають вміст безпосередньо на сторінці. Інші теги, такі як <p>, оточують текст і надають інформацію про нього, і можуть містити інші теги як дочірні елементи. Замість відображення тегів HTML браузери використовують їх для інтерпретації вмісту сторінки [22].

У HTML можна вбудовувати програми, написані на скриптових мовах, наприклад JavaScript, які впливають на поведінку та вміст WEB-сторінок [23].

HTML впроваджує засоби для [23]:

- створення структурованого документа шляхом позначення структурного складу тексту (заголовки, абзаци, списки, таблиці, цитати та інше);
- отримання інформації із мережі інтернет через гіперпосилання;
- створення інтерактивних форм;
- включення зображень, звуку, відео, та інших об’єктів до тексту.

Включення CSS визначає зовнішній вигляд і макет вмісту. Консорціум World Wide Web (W3C), який контролює стандарти HTML і CSS, заохочує використання CSS замість явного HTML з 1997 року [23].

CSS (каскадні таблиці стилів) – це особливий стиль мови сторінок, який використовується для опису їх зовнішнього вигляду. Самі сторінки написані мовою розмітки даних [24].

CSS є основною технологією у WWW (World Wide Web) разом із HTML і JavaScript. CSS найчастіше використовується для візуального відтворення

сторінок, написаних у HTML і XHTML, але форматування CSS також можна застосовувати до інших типів XML-документів. Специфікація CSS була створена World Wide Web Consortium і знаходиться на стадії розробки. CSS має різні рівні та профілі. CSS наступного рівня будується на основі попереднього CSS, додаючи нову функціональність або розширюючи існуючу. Рівні представлені як CSS1, CSS2 і CSS3. Профіль – це набір правил CSS на одному чи кількох рівнях, створених для певного типу пристрою чи інтерфейсу. Наприклад, існують профілі CSS для принтерів, мобільних пристроїв тощо [25].

React – це бібліотека JavaScript з відкритим вихідним кодом для створення інтерфейсів користувача, призначена для вирішення проблеми часткового оновлення вмісту WEB-сторінки, яка виникає під час розробки односторінкових програм. Розроблено Meta та індивідуальною спільнотою розробників [26].

React дозволяє розробникам створювати великі WEB-застосунки, які використовують дані, котрі змінюються з часом, без перезавантаження сторінки. Його мета полягає в тому, щоб бути швидким, простим, масштабованим. React обробляє тільки користувацький інтерфейс у застосунках. Властивості передаються в рендерер компоненту, як властивості html тегу. Компонент не може напряму змінювати властивості, що йому передані, але може їх змінювати через callback функції. Такий механізм називають «властивості донизу, події нагору». Це відповідає видові у шаблоні модель-вид-контролер (MVC), і може бути використане у поєднанні з іншими JavaScript бібліотеками або в великих фреймворках MVC, таких як AngularJS. Він також може бути використаний з React на основі надбудов, щоб піклуватися про частини без користувацького інтерфейсу побудови WEB-застосунків [27].

3.2 Розробка клієнтської частини системи автоматизованого тестування WEB-додатків

На даний момент розробка WEB-додатку із використанням лише HTML та CSS вважається архаїчним методом проектування. Навіть при використанні додаткового інструментарію, як XML-документи чи PHP, все ще залишаються застарілими, через надлишкові звертання до сервера, що може знизити швидкість відповіді до користувача. Тому більш доцільне проектування з використанням single-page application.

Односторінковий застосунок, або односторінковий інтерфейс – це WEB-застосунок чи WEB-сайт, який вміщується на одній сторінці з метою забезпечити користувачу досвід близький до користування настільною програмою. В односторінковому застосунку весь необхідний код - HTML, JavaScript, та CSS - завантажується разом зі сторінкою, або динамічно довантажується за потребою, зазвичай у відповідь на дії користувача. Сторінка не оновлюється і не перенаправляє користувача до іншої сторінки у процесі роботи з нею. Взаємодія з односторінковим застосунком часто включає в себе динамічний зв'язок з WEB-сервером [28].

Структурна схема, що зображена на рисунку 3.1, демонструє зв’язки між модулями, шляхами у WEB-додатку.

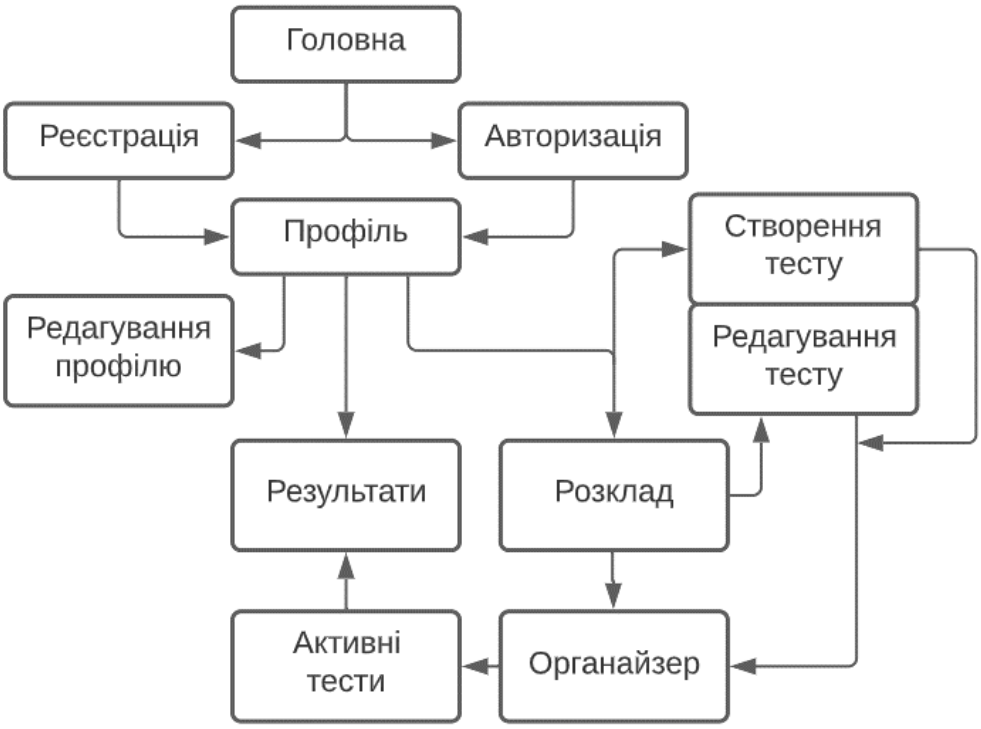


Рисунок 3.1 – Структурна схема архітектури клієнтської частини системи автоматизованого тестування WEB-додатків

Бібліотека, що дозволяє створювати односторінкові застосунки, це React. Першим кроком, під час використання React, потрібно завантажити порожній проект-шаблон, в якому буде автоматично створена база, для подальшої розробки. Для цього виконаємо наступні команди у директорії, в якій буде розташовуватись даний проєкт:

```
npx create-react-app my-app
cd my-app
npm start.
```

Завантажувальний шаблон містить базові зв’язки HTML документа та JavaScript скриптів, за допомогою яких і будуть описуватись ті чи інші елементи

на сторінці. Для початку потрібно описати шляхи, за якими буде відбуватись переміщення по сторінкам даного WEB-додатку.

У React для опису шляхів потрібно створити окремий елемент, який називається маршрутизатор – «Router»:

```
export default function App() {
  return (
    <Router>
      <div>
        <ul>
          <li>
            <Link to="/">Home</Link>
          </li>
          <li>
            <Link to="/about">About</Link>
          </li>
          <li>
            <Link to="/users">Users</Link>
          </li>
        </ul>.
```

Алгоритм роботи розробленого методу автоматизованого тестування WEB-додатків можна умовно розділити на дві частини: одна з яких буде відповідати за взаємодію з користувачем – авторизація, створення черги тестів, збереження даних про тест та виведення результатів тестування користувачу; а інша – за проведення тестування, відповідно у цій частині будуть почергово обиратись тести та відповідно отримуватись інформація про те чи інше значення вимірювання (рис. 3.2).

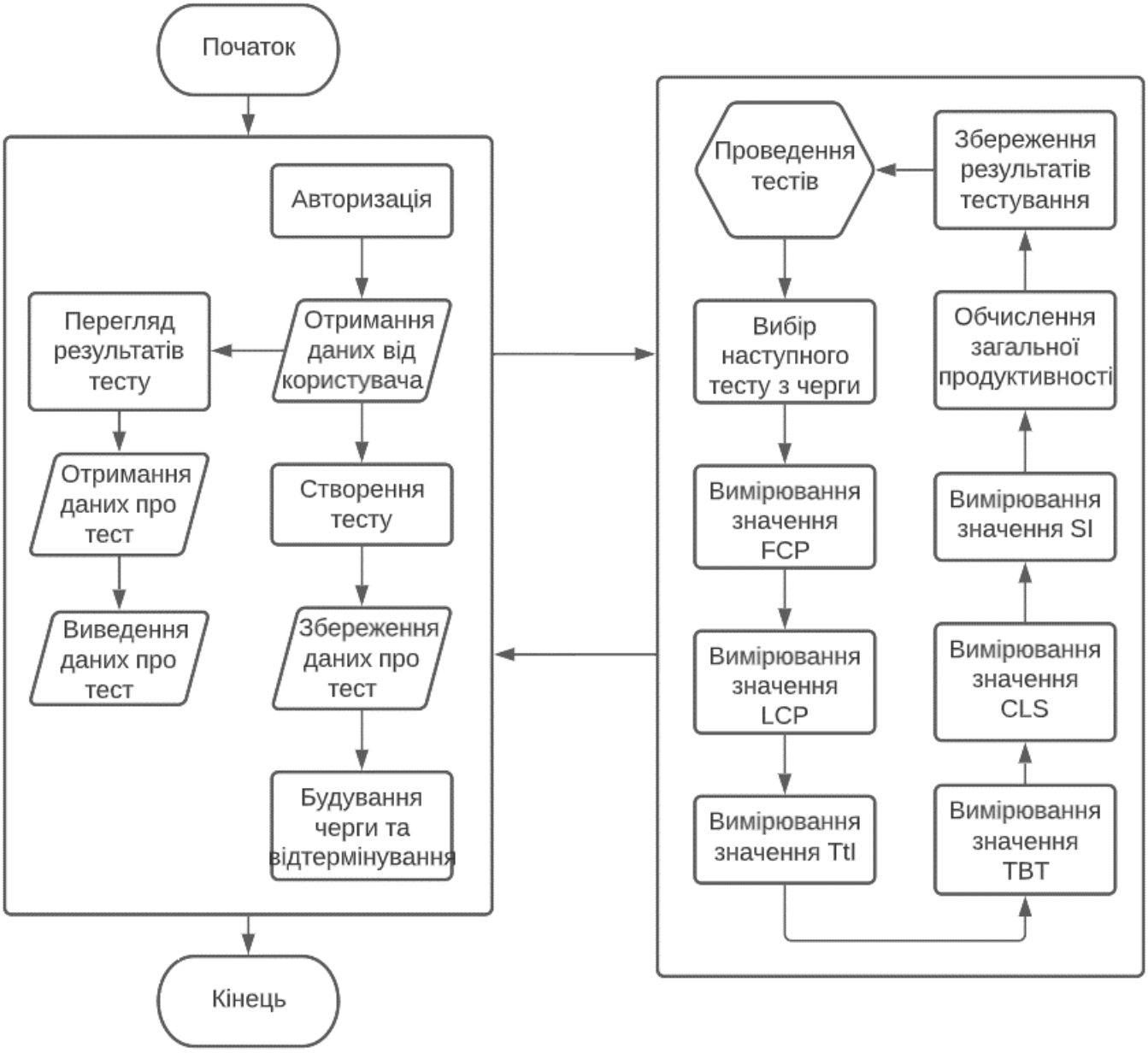


Рисунок 3.2 – Алгоритм роботи розробленого методу автоматизованого тестування WEB-додатків

Усього таких значень шість – час початкового відображення вмісту (FCP), час завантаження WEB-сторінки (SI), час відображення найбільшої частини вмісту (LCP), час до взаємодії зі сторінкою (TtI), загальний час блокування WEB-сторінки (TBT), кумулятивний зсув макета (LCS).

У цьому прикладі маршрутизатор обробляє три «сторінки»: домашню сторінку, сторінку з інформацією та сторінку користувача. Коли натискаються різні посилання <Link>, маршрутизатор відтворює відповідний <Route>.

```
      <Switch>
        <Route path="/about">
          <About />
        </Route>
        <Route path="/users">
          <Users />
        </Route>
        <Route path="/">
          <Home />
        </Route>
      </Switch>
    </div>
  </Router>
);.
```

<Switch> переглядає свої дочірні елементи <Route> включно з першим, який відповідає поточній URL-адресі.

Відповідно далі потрібно налаштувати кожну з цих адрес, на які посилається маршрутизатор. Для збереження чистоти коду, рекомендується створювати елементи, що відповідають за різні функції у різних файлах. А самі елементи можуть описуватись різними шляхами.

Наприклад як функція що експортується:

```
export default function functionToExportExample() {
   return (
      <div className="Class name example">
         Some text or other info here!
      </div>
   );
}.
```

Або як callback який зберігається у константі:

```
const constObjectExample = ({}) => {
   return (
      <TagNameExample>
         Some text or other info here!
      < TagNameExample />
   );
};
export default CreateFact;.
```

Між даними прикладами різниці майже немає, тому є можливим використання обох розглянутих прикладів.

Кожен елемент, незалежно від методу його опису, як результат повинен повертати JSX-типовий об’єкт. JSX – це розширення синтаксису для JavaScript. JSX рекомендовано використовувати в React, щоб описати, як повинен виглядати інтерфейс користувача. JSX може нагадувати мову шаблонів, але з усіма перевагами JavaScript. JSX створює “React-елементи”, які у свою чергу використовуються підчас рендерингу WEB-сторінки.

Сторінка з результатами тестування повинна містити інформацію проведених тестів, використовуючи для наглядності кольорову ідентифікацію та числові значення, що мають привернути увагу користувача про ті чи інші проблеми із WEB-додатком. Схематичних вигляд карток, що містить дану інформацію зображений на рисунку 3.3.

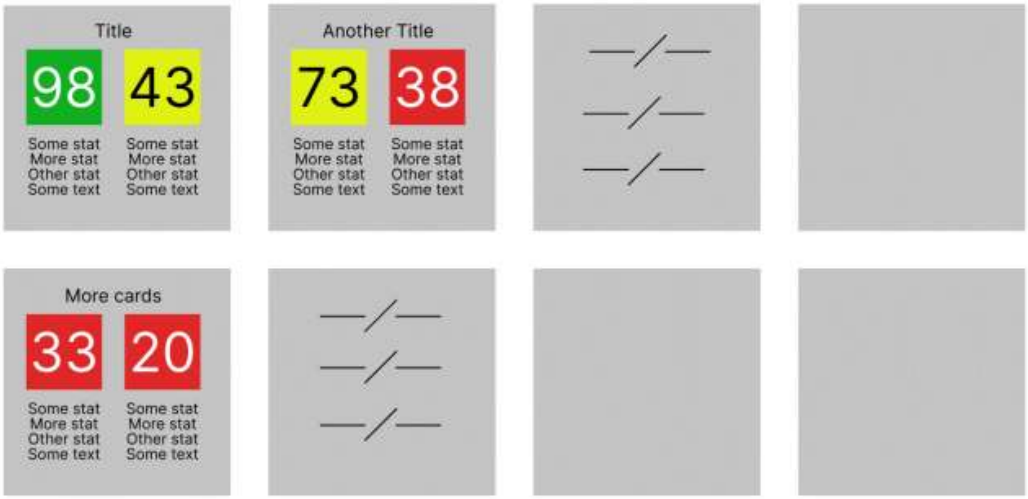


Рисунок 3.3 – Схема інтерфейсу головної сторінки системи автоматизованого тестування WEB-додатків

Продовжуючи розробку сторінок, опишемо як повинна виглядати картка, та список карток з результатами тестування.

```
const Card = ({data}) => {
   return (
      <div className="card_base">
         <div className="col s12 m3">
            <div className="card">
               <div className="card-image">
                  <img src={data.cardImg} />
                  <span className="card-title">{data.cardTitle}</span>
               </div>
               <div className="card-content">
                  <div>{data.cardText}</div>
               </div>
               <div className="card-action">
                  <a href="#">X</a>
               </div>
            </div>
         </div>
      </div>
   );.
```

Вищеописаний об’єкт являє собою блок розмірами 100% від розміру його батьківського елементу в ширину, та 33% від розміру екрану користувача у висоту. Містить зображення, заголовок та інформацію, що буде отримана при подальшому об’явленні даної картки.

```
const CardsList = ({data}) => {
   return (
      <div className="card_list row">
         {data.map((cardData, index) => {
            return <Card data={cardData} key={index}/>
         })}
      </div>
   )};.
```

Наступним елементом буде список карток. Цей елемент приймає масив даних про усі картки, та нарізає їх відповідно у кожен «Card» елемент.

Так як, даний WEB-додаток, що розробляється, розділений на клієнтську та серверну частини, тому для отримання інформації від сервера, потрібно зв’язати клієнт та сервер. Для цієї цілі, і навіть більше, у React існують «хуки».

«Хуки» – це новинка в React 16.8. Вони дозволяють використовувати стан та інші можливості React без написання класу. Всього «хуків» у React є 10, з них базові [29]:

- useState – Повертає значення стану та функцію, що оновлює його. Функція використовується для оновлення стану. Вона приймає значення нового стану і ставить у чергу повторний рендер компонента.
- useEffect – Приймає функцію, що містить імперативний, можливо з ефектами, код. Функція, передана в useEffect, буде запущена після того, як вивід рендеру з’явиться на екрані.

– useContext – Приймає об’єкт контексту і повертає поточне значення контексту для нього. Поточне значення контексту визначається прапором value найближчого провайдера, що знаходиться вище у дереві компонентів.

Та додаткові «хуки»: useReducer, useCallback, useMemo, useRef, useImperativeHandle, useLayoutEffect, useDebugValue.

Відповідно, для зв’язування клієнта з сервером, можна використати існуючий «хук» «useState», при створенні нового «хука», для зв’язку з даними сервера.

```
export const useFetching = (callback) => {
   const [isLoading, setIsLoading] = useState(false);
   const [error, setError] = useState('');
   const fetching = async (...args) => {
      try {
         setIsLoading(true)
         await callback(...args)
      } catch (e) {
         setError(e.message);
      } finally {
         setIsLoading(false)
      }
   }
   return [fetching, isLoading, error]
}.
```

3.3 Тестування клієнтської частини системи автоматизованого тестування WEB-додатків

Для проведення тестування клієнтської частини, першим кроком потрібно запустити збудований React-проект у локальній мережі. Відповідно ввівши у командний рядок, у контексті проекту, команду – npm start.

Після вводу команди термінал виведе повідомлення, що зображене на рисунку 3.4.

```
Starting the development server...

```

Рисунок 3.4 – Повідомлення про початок процесу завантаження проекту у локальній мережі

Через деякий час (5-7 хвилин) проект повноцінно завантажиться, та автоматично буде відкрите посилання у локальній мережі (рис. 3.5).

Рисунок 3.5 – Сторінка авторизації інформаційної технології автоматизованого тестування WEB-додатків

Наступним кроком потрібно зареєструвати користувача у системі, відповідно ввівши дані у форму реєстрації (рис. 3.6).

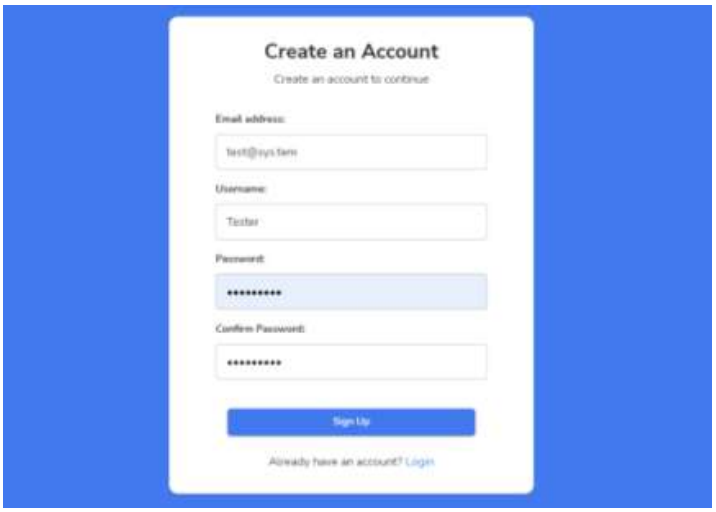


Рисунок 3.6 – Заповнена форма реєстрації користувача інформаційної технології автоматизованого тестування WEB-додатків

Після реєстрації відбудеться перенаправлення на головну сторінку, для нового користувача вона порожня з підказкою, що черга тестів порожня, але для користувача із заповненим розкладом тестувань, головна сторінка буде виглядати як зображено на рисунку 3.7.

Рисунок 3.7 – Загальний вигляд головної сторінки програного модуля клієнтської частини автоматизованого тестування WEB-додатків із заповненим розкладом

Для запису відповідного тесту до черги, потрібно заповнити форму на сторінці «Producers» (рис. 3.9).

Поле «Title» відповідає за назву тесту, під введеною назвою даний тест буде збережений у черзі. «Endpoint» – відповідає за пряме посилання на WEB-додаток який потрібно протестувати. «Producer Type» – вид тесту, різні види тестів містять різні характеристики що потребує користувач. «Producer Frequency» – частота проведення тестів, є всього чотири частоти: щогодини, щодня, щотижня, щомісяця.

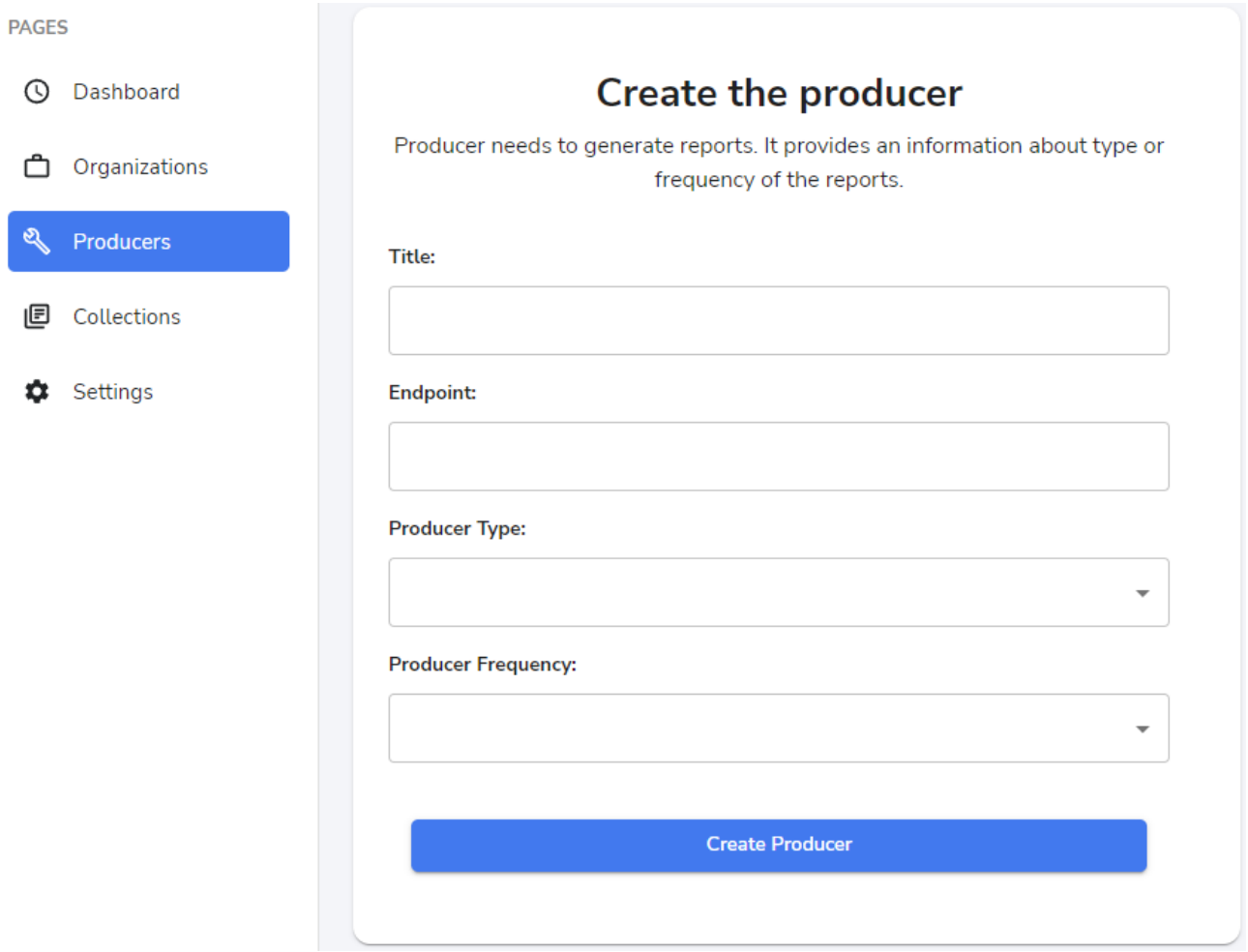


Рисунок 3.8 – Форма запису тесту до черги у інформаційній технології автоматизованого тестування WEB-додатків

Тест записаний у розклад буде відображатись на сторінці «Collections». На цій сторінці відображається коротка, згрупована інформація про проведені тестування. Щоб отримати детальний опис одного конкретного тесту, його можна відкрити, обравши результати за потрібну дату (рис. 3.9).

Рисунок 3.9 – Загальний вигляд прикладу звіту проведеного тестування у інформаційній технології автоматизованого тестування WEB-додатків

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

Час початкового відображення вмісту – 720 мс; час завантаження WEB-сторінки – 820 мс; час відображення найбільшої частини вмісту – 820 мс; час до взаємодії зі сторінкою – 780 мс; загальний час блокування WEB-сторінки – 50 мс; кумулятивний зсув макета (CLS) – 0,0 ум.од.

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 (час початкового відображення вмісту – 720 мс; час завантаження WEB-сторінки – 820 мс; час відображення найбільшої частини вмісту – 820 мс; час до взаємодії зі сторінкою – 780 мс; загальний час блокування WEB-сторінки – 50 мс; кумулятивний зсув макета (CLS) – 0,0 ум. од.) розробленої інформаційної технології автоматизованого тестування WEB-додатків, є вищим, а саме у 1,32 рази порівняно з JMeter, у 1,34 рази порівняно з TestCafe та у 3,23 рази порівняно із ODT, що доводить ефективність розробленої інформаційної технології.

4 ЕКОНОМІЧНА ЧАСТИНА

4.1 Комерційний та технологічний аудит науково-технічної розробки

Метою проведення комерційного і технологічного аудиту є оцінювання науково-технічного рівня та рівня комерційного потенціалу інформаційної технології автоматизованого тестування WEB-додатків. Клієнтська частина.

Для проведення комерційного та технологічного аудиту залучають не менше трьох 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 табл. 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 працездатність продукту в реальних
Ринкові переваги					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому	Один аналог на великому ринку	Продукт не має аналогів на великому

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 ціни аналогів
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 аналогів
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 аналогів
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

Продовження табл. 4.1

9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування	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 використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти	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	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

Як видно з таблиці, рівень комерційного потенціалу розроблюваного нового програмного продукту є високим та становить 44 бали, що досягається за рахунок того, що інформаційна модель відрізняється від існуючих використанням розширеного функціоналу при автоматизації тестування WEB-додатків та використання інтелектуального аналізу даних та оптимізації використання ресурсів при роботі програмного забезпечення, що дозволить підвищити загальну швидкодію роботи WEB-додатків.

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.

$$q_i = \frac{800}{700} = 1.11$$

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

Параметр	Аналог	Нова розробка	Індекс зміни значення параметра	Коефіцієнт вагомості
Технічні				
Час початкового відображення вмісту (мс)	800	720	1,11	0,25
Час відображення найбільшої частини вмісту (мс)	1 500	820	1,83	0,11
Час до взаємодії зі сторінкою (мс)	980	780	1,26	0,19
Загальний час блокування WEB-сторінки (мс)	80	50	1,60	0,24
Кумулятивний зсув макета (ум. од.)	0,015	0	-	-
Час завантаження WEB-сторінки (мс)	900	820	1,10	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:

$$I_{НП} = \prod_{i=1}^{n} q_i,$$

де $I_{НП}$ – 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 $I_{НП} = 1$.

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

$$I_{ТП} = \sum_{i=1}^{n} q_i \cdot \alpha_i,$$

де $I_{тп}$ – 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.

$$I_{ТП} = 1.11 * 0.25 + 1.83 * 0.11 + 1.26 * 0.19 + 1.6 * 0.24 + 1.1 * 0.2 = 1.34$$

Груповий параметричний індекс за економічними параметрами розраховуємо за формулою:

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

де $I_{\text{ЕП}}$ – 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.

$$\mathrm{I_{ЕП}} = 0{,}53 * 0{,}4 + 0{,}67 * 0{,}6 = 0{,}61.$$

Етап 3. Розрахунок інтегрального показника.

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

$$\mathrm{К_{ІНТ}} = \mathrm{I_{НП}} \cdot \frac{\mathrm{I_{ТП}}}{\mathrm{I_{ЕП}}}$$

$$\mathrm{К_{ІНТ}} = 1 * \frac{1{,}34}{0{,}61} = 2{,}19.$$

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 дослідників, яка розраховується за формулою:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \times t_i}{T_p}, \quad (4.1)$$

де k – кількість посад дослідників, залучених до процесу розробки; Мпі – місячний посадовий оклад конкретного дослідника, грн; ti – кількість днів роботи конкретного дослідника, дн.; Тр – середня кількість робочих днів в місяці, Тр=21…23 дні.

Результати розрахунків зведемо до таблиці 4.4.

Таблиця 4.4 – Основна заробітна плата дослідників

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник проекту	25000	1136,36	80	90 909,09
Інженер	21000	954,55	80	76 363,64
Всього				167 272,73

Так як в даному випадку розробляється програмний продукт, то дослідник 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:

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

$$З_д = \left(167\ 272{,}73 \cdot \frac{11\%}{100\%}\right) = 18\ 400{,}00 \text{ (грн.)}.$$

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

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

$$H_з = (167\ 272{,}73 + 18\ 400{,}00) \cdot \frac{22\%}{100\%} = 40\ 848{,}00 \text{ (грн.)}.$$

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:

$$A = \frac{Ц}{Тв} \cdot \frac{t_{вик}}{12}, \quad (4.4)$$

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

Розрахуємо, для прикладу, амортизаційні витрати на комп’ютер балансова вартість якого становить 17200 грн., термін його корисного використання згідно податкового законодавства – 2 роки, а термін його фактичного використання – 4 місяці:

$$А_{обл} = \frac{17200}{2} \times \frac{4}{12} = 2\ 866{,}67 \text{ грн.}$$

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

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

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	Амортиза-ційні відрахуван-ня, грн.
Комп'ютер	17 200,00	2,00	4,00	2 866,67
Приміщення	500 000,00	20,00	4,00	8 333,33
Сервер	30 000,00	2,00	4,00	5 000,00
Ліцензійна ОС, та спеціалізовані ліцензійні нематеріальні ресурси	13 500,00	-	4,00	450,0
Всього				17 100,00

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:

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

де W_{yi} – встановлена потужність обладнання на певному етапі розробки, кВт; t_i – тривалість роботи обладнання на етапі дослідження, год; $Ц_е$ – 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.

$$Ц_е = \left(Ц_{опт} + Ц_{розп} + Ц_{пост}\right)\left(1 + \frac{ПДВ}{100\%}\right), \quad (4.7)$$

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

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 80 повних робочих дні – 640 годин, а коефіцієнт, що враховує використання потужності – 0,8 кВт/год

$$B_{ек} = W_{yi} \times t_i \times Ц_е \times К_{впі} = 0{,}09 * 640 * 6{,}02 * 0{,}8 = 277{,}40 \text{ грн.}$$

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
Комп'ютер	0,09	80*8=640,00	277,40
Сервер	0,12	80*24=1920,00	1 151,22
Приміщення	0,029	80*8=640	106,14
Всього			1 534,76

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:

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

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

$$\text{І}_\text{в} = (167\ 272{,}73 + 0) * \frac{115\%}{100\%} = 192\ 363{,}64 \text{ (грн.)}.$$

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:

$$\text{Н}_\text{нзв} = \left(\text{З}_\text{о} + \text{З}_\text{р}\right) \cdot \frac{\text{Н}_\text{нзв}}{100\%}, \qquad (4.9)$$

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

$$\text{Н}_\text{нзв} = (167\ 272{,}73 + 0) * \frac{126\%}{100\%} = 210\ 763{,}64 \text{ (грн.)}.$$

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

$$\text{В}_\text{заг} = 167\ 272{,}73 + 18\ 400{,}00 + 40\ 848{,}00 + 17\ 100{,}00 + 1\ 534{,}76 + 192\ 363{,}64 + 210\ 763{,}64 = 631\ 182{,}76 \text{ грн.}$$

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:

$$ЗВ = \frac{631\,182{,}76}{0{,}5} = 1\,262\,365{,}53 \text{ грн.}$$

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; $Ц_0$ – 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:

$$\Delta П_1 = (500 * 0 + (1\,000 + 500) * 10\,000) * 0{,}8333 * 0{,}45 * (1 - 0{,}18)$$
$$= 4\,612\,315{,}50 \text{ грн.}$$

$$\Delta П_2 = \left(500 * 0 + (1\,000 + 500) * (10\,000 + 15\,000)\right) * 0{,}8333 * 0{,}45 * (1 - 0{,}18) = 11\,530\,788{,}75 \text{ грн.}$$

$$\Delta П_3 = \left(500 * 0 + (1\,000 + 500) * (10\,000 + 15\,000 + 20\,000)\right) * 0{,}8333 * 0{,}45 * (1 - 0{,}18) = 20\,755\,419{,}75 \text{ грн.}$$

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:

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

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:

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

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

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

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:

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

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

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 відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $Е_в$. Для цього використаємо формулу:

$$E_в = \sqrt[Т_ж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (4.15)$$

де $T_ж$ –життєвий цикл наукової розробки, час від початку її розробки до закінчення отримування позитивних результатів від її впровадження роки.

$$E_в = \sqrt[з]{1 + \frac{26\,791\,714{,}58}{2\,524\,731{,}05}} - 1 = 1{,}264$$

Визначимо мінімальну ставку дисконтування, за формулою:

$$\tau_{мін} = d + f, \quad (4.16)$$

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).

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

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

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$\mathrm{T_{ок}} = \frac{1}{\mathrm{E_в}}, \tag{4.17}$$

$$\mathrm{T_{ок}} = \frac{1}{1.264} = 0{,}791 \text{ р.}$$

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 – 1 262 365,53 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. В результаті аналізу розрахунків можна зробити висновок, що розроблений програмний продукт є високо конкурентоспроможним та період окупності складе близько 0,791 роки.

ВИСНОВКИ

У результаті 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 клієнтських модулів програмних систем автоматизованого тестування WEB-додатків та порівняльного аналізу характеристик клієнтських модулів програмних систем автоматизованого тестування WEB-додатків.

Було здійснено проектування інтелектуальної моделі системи автоматизованого тестування WEB-додатків, розроблено математичну модель динамічного автоматизованого тестування WEB-додатків та метод автоматизованого динамічного тестування WEB-додатків, проведено проектування інтелектуальної моделі системи автоматизованого тестування WEB-додатків;

Програмно реалізовано систему автоматичного тестування WEB-додатків на клієнтському рівні. Обґрунтовано вибір інструментарію при розробці клієнтської частини системи автоматизованого тестування WEB-додатків, проведено тестування клієнтської частини системи автоматизованого тестування WEB-додатків. У результаті тестування визначенні характеристики розробленої програми, а саме: час початкового відображення вмісту – 720 мс; час завантаження WEB-сторінки – 820 мс; час відображення найбільшої частини вмісту – 820 мс; час до взаємодії зі сторінкою – 780 мс; загальний час блокування WEB-сторінки – 50 мс; кумулятивний зсув макета (CLS) – 0,0. У порівнянні з аналогом «JMeter» – час початкового відображення вмісту – 800 мс; час відображення найбільшої частини вмісту – 1500 мс; час до взаємодії зі сторінкою – 980 мс; загальний час блокування WEB-сторінки – 80 мс; кумулятивний зсув

макета – 0,015; час завантаження WEB-сторінки – 900 мс. Що загалом дали результати збільшення ефективності у щонайменше 1,32 рази.

Виконано розрахунок витрат на розробку нового програмного продукту, сума яких складає 1 262 365,53 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 0,791 роки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Я. В. Іванчук, В. Р. Затковський Інформаційна технологія автоматичного тестування веб-додатків. Клієнтська 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-fksa-2022/paper/view/15680.
2. Шпаргалка з тестування [Електронний ресурс]. Режим доступу – https://qalearning.com.ua/theory/about_qa/shpargalka-z-testuvannya/.
3. Що таке WEB-додаток? [Електронний ресурс]. Режим доступу – https://ukr.4meahc.com/what-exactly-is-web-application-50384.
4. What is Manual Testing? [Електронний ресурс]. Режим доступу – http://www.softwaretestingclass.com/what-is-manual-testing/.
5. Practical Experience in Automated Software Testing [Електронний ресурс]. Режим доступу – https://www.methodsandtools.com/archive/ archive.php?id=3.
6. When Should a Test Be Automated? [Електронний ресурс]. Режим доступу – https://www.stickyminds.com/article/when-should-test-be-automated.
7. Особливості web-додатків [Електронний ресурс]. Режим доступу – http://sites.znu.edu.ua/webprog/lect/1191.ukr.html.
8. Тестування WEB-проєктів: основні етапи та поради [Електронний ресурс]. Режим доступу – https://qalight.ua/baza-znaniy/testuvannya-veb-proєktiv-osnovni-etapi-ta-poradi/.
9. Apache JMeter™ [Електронний ресурс]. Режим доступу – https://jmeter.apache.org.

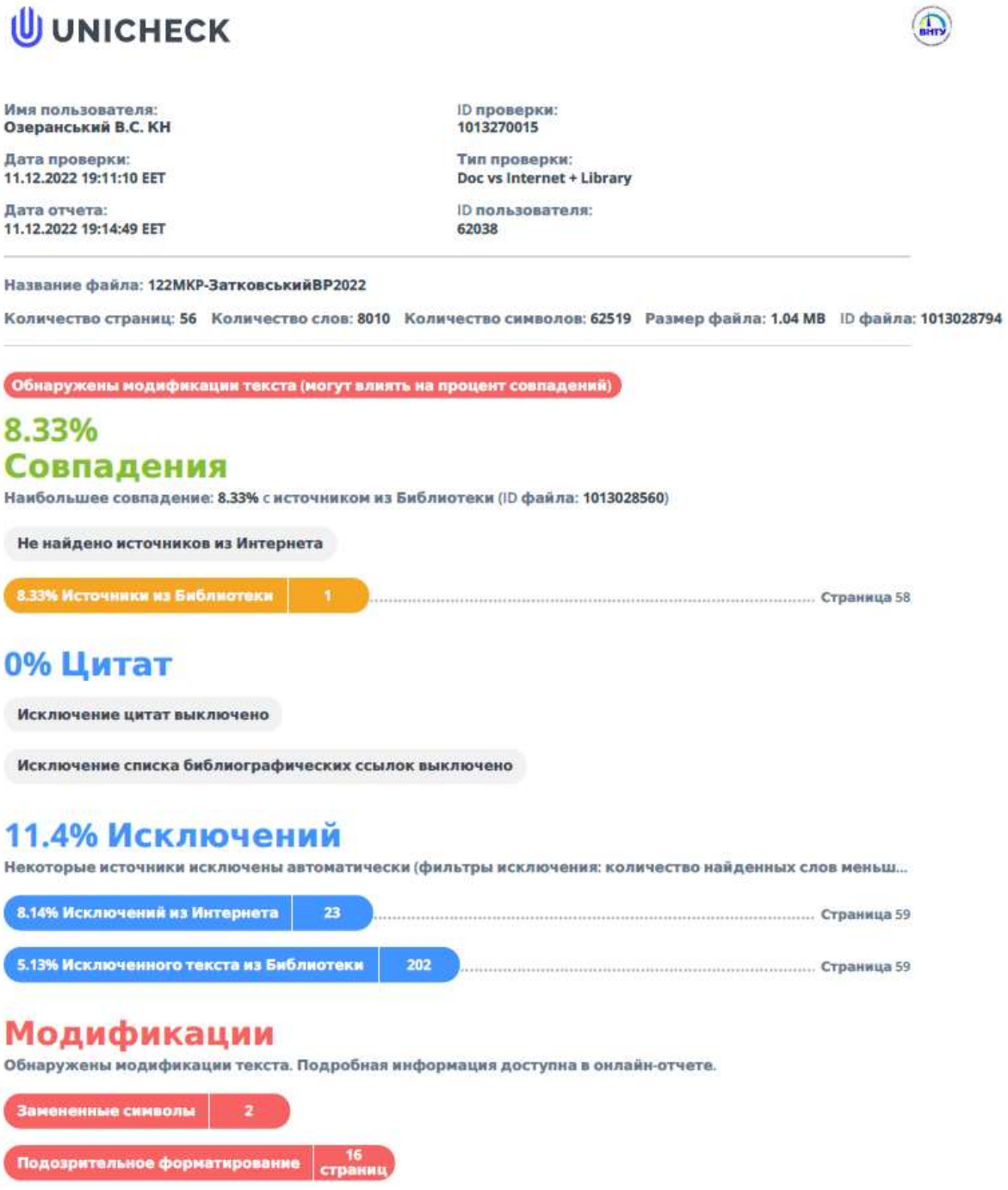
10. Difference Between Object-Oriented Testing and Conventional Testing [Електронний ресурс]. Режим доступу – https://www.geeksforgeeks.org/difference-between-object-oriented-testing-and-conventional-testing/.
11. Object-Driven Testing - Basic Concepts [Електронний ресурс]. Режим доступу – https://support.smartbear.com/testcomplete/docs/testing-with/deprecated/odt/basic-concepts.html.
12. Why people love TestCafe [Електронний ресурс]. Режим доступу – https://testcafe.io.
13. Why TestCafe? [Електронний ресурс]. Режим доступу – https://testcafe.io/documentation/402631/why-testcafe.
14. TestCafé Studio [Електронний ресурс]. Режим доступу – https://www.devexpress.com/products/testcafestudio/.
15. The way of the web tester / Jonathan Rasmusson – M. Pragmatic Bookshelf;, 2016. – 258 c.
16. Статична і динамічна методики тестування [Електронний ресурс]. Режим доступу – https://training.qatestlab.com/blog/technical-articles/static-and-dynamic-testing-methods/.
17. Статичне та динамічне тестування [Електронний ресурс]. Режим доступу – https://qalight.ua/baza-znaniy/ctatichne-ta-dinamichne-testuvannya/.
18. What is Dynamic Testing? [Електронний ресурс]. Режим доступу – https://www.guru99.com/dynamic-testing.html.
19. R. Alur, M. Benedikt, K. Etessami, P. Godefroid, T. Reps, and M. Yannakakis. Analysis of Recursive State Machines. TOPLAS, 27(4):786–818, July 2005 – 818 p.
20. R. Alur and M. Yannakakis. Model Checking of Hierarchical State Machines. In FSE’98 – 203 p.

21. K. Sen, D. Marinov, and G. Agha. CUTE: A Concolic Unit Testing Engine for C. In FSE’2005 - 123 p.
22. Найкращі інструменти WEB-розробки [Електронний ресурс]. Режим доступу – https://uk.myservername.com/top-13-best-front-end-web-development-tools-consider-2021.
23. Підручники HTML і CSS [Електронний ресурс]. Режим доступу – https://htmlbook.at.ua.
24. HTML & CSS [Електронний ресурс]. Режим доступу – https://web.archive.org/web/20101129081921/http://www.w3.org/standards/webdesign/htmlcss.
25. Довідник по HTML тегам [Електронний ресурс]. Режим доступу – https://css.in.ua/html/tags.
26. Довідник по CSS властивостям [Електронний ресурс]. Режим доступу – https://css.in.ua/css/properties.
27. React [Електронний ресурс]. Режим доступу – https://uk.reactjs.org
28. Facebook-React v15.3.1 [Електронний ресурс]. Режим доступу – https://web.archive.org/web/20160914010018/https://github.com/facebook/react/releases.
29. Inner-browsing extending the browser navigation paradigm [Електронний ресурс]. Режим доступу – https://web.archive.org/web/20140327035127/https://developer.mozilla.org/en-US/docs/Inner-browsing_extending_the_browser_navigation_paradigm.
30. API-довідник хуків [Електронний ресурс]. Режим доступу – https://uk.reactjs.org/docs/hooks-reference.html.
31. 16.09.2021 ОГОЛОШЕННЯ [Електронний ресурс]. Режим доступу – https://epvmvntu.wordpress.com/2021/09/16/.

Додаток А

(обов'язковий).

Результат перевірки на антиплагіат у системі «UNICHECK»



UNICHECK

Имя пользователя:
Озеранський В.С. КН

ID проверки:
1013270015

Дата проверки:
11.12.2022 19:11:10 EET

Тип проверки:
Doc vs Internet + Library

Дата отчета:
11.12.2022 19:14:49 EET

ID пользователя:
62038

Название файла: 122МКР-ЗатковськийВР2022

Количество страниц: 56 Количество слов: 8010 Количество символов: 62519 Размер файла: 1.04 MB ID файла: 1013028794

Обнаружены модификации текста (могут влиять на процент совпадений)

8.33%
Совпадения

Наибольшее совпадение: 8.33% с источником из Библиотеки (ID файла: 1013028560)

Не найдено источников из Интернета

8.33% Источники из Библиотеки | 1 ..... Страница 58

0% Цитат

Исключение цитат выключено

Исключение списка библиографических ссылок выключено

11.4% Исключений

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

8.14% Исключений из Интернета | 23 ..... Страница 59

5.13% Исключенного текста из Библиотеки | 202 ..... Страница 59

Модификации

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

Замененные символы | 2

Подозрительное форматирование | 16 страниц

Додаток Б

(обов’язковий)

Лістинг програми

```
export type AppRouteProps = {
   accessType?: AppRouteAccessType;
   roles?: UserRole[];
} & RouteProps;

export const APP_ROUTES: AppRouteProps[] = [
   {
      path: '/',
      component: AuthView,
      accessType: AppRouteAccessType.Guest,
      exact: true
   },
   {
      path: '/test',
      component: TestPage,
      accessType: AppRouteAccessType.Authorized,
      exact: true
   },
   {
      path: '/dashboard',
      component: DashboardView,
      accessType: AppRouteAccessType.Authorized,
   },
   {
```

```
        path: '/producers',
        component: ProducersView,
        accessType: AppRouteAccessType.Authorized,
        roles: [UserRole.Admin, UserRole.Owner]
    },
    {
        path: '/organizations',
        component: OrganizationView,
        accessType: AppRouteAccessType.Authorized,
    },
    {
        path: '/collections',
        component: CollectionsView,
        accessType: AppRouteAccessType.Authorized
    },
    {
        path: '/settings',
        component: SettingsView,
        accessType: AppRouteAccessType.Authorized
    }
];

export const AUTHORIZATION_PATH = '/';

export const AUTHORIZED_START_PATH = '/dashboard';

const useStyles = makeStyles(theme => ({
    root: {
```

```
        display: 'flex'
    },
    content: {
        flex: 1,
        padding: theme.spacing(3)
    },
    toolbar: theme.mixins.toolbar
}));

export const DefaultLayout: React.FC = ({children}) => {
    const {state} = useUserState();
    const classes = useStyles();

    if (!state.currentUser) {
        return <> {children} </>;
    }

    return <div className={classes.root}>
        <Navbar />
        <AppDrawer />

        <div className={classes.content}>
            <div className={classes.toolbar} />

            {children}
        </div>
    </div>
}
```

```
export const LighthouseReportsChart: React.FC<Props> = (props) => {
  const { data } = props;
  const theme = useTheme();

  return (
    <>
      <ResponsiveContainer width="100%" height={300}>
        <BarChart
          width={500}
          height={300}
          data={data}
          margin={{
            top: 5,
            right: 30,
            left: 20,
            bottom: 5,
          }}
          reverseStackOrder={false}
        >
          <CartesianGrid strokeDasharray="3 3" />
          <XAxis dataKey="name" />
          <YAxis />
          <Tooltip />
          <Legend />
          <Bar dataKey="mobile" fill={theme.palette.primary.main} />
          <Bar dataKey="desktop" fill={theme.palette.secondary.main} />
        </BarChart>
```

```
            </ResponsiveContainer>
          </>
        );
      };

      export const Navbar = () => {
        const { state } = useUserState();
        const { state: producersState, dispatch } = useProducersState();

        const classes = useStyles();

        const changeProducer = (id: string) => {
          dispatch(new SelectProducerAction(id));
        }

        return <AppBar position={'fixed'} className={classes.root}>
          <Toolbar>
            <div className={classes.selectWrapper}>
              { producersState.producers.length > 0 && <NavbarProducersSelect
                producers={producersState.producers}
                currentProducerId={producersState.currentProducer?.id as
string}
                onChange={changeProducer}
              /> }
            </div>
            { state.currentUser && <NavbarProfile
              currentUser={state.currentUser}
              currentRole={state.currentRole}
```

```
            /> }
        </Toolbar>
    </AppBar>
}

export const StatsCard = (props: Props) => {
    const {
        title = '',
        value = '',
        variant = 'purple',
        description = '',
        icon
    } = props;

    const classes = useStyles({
        variant: variant
    });

    return <Paper>
        <Box p={2}>
            <Grid container>
                <Grid item xs={9}>
                    <Box m={1} />
                    <Typography variant={'h4'} className={classes.title}>
                        { title }
                    </Typography>
                    <Box m={1} />
                    <Typography variant={'h2'}>
```

```
                        { value }
                    </Typography>
                </Grid>
                <Grid item xs={3}>
                    <Box display={'flex'} justifyContent={'center'} alignItems={'center'} className={classes.icon}>
                        {icon && icon()}
                    </Box>
                </Grid>
            </Grid>
            <Box m={3} />
            <Typography variant={'caption'} className={classes.description}>
                { description }
            </Typography>
        </Box>
    </Paper>
}

export const AuthLoginForm = () => {
    const { dispatch } = useUserState();

    const form = useForm({
        resolver: yupResolver(schema)
    });

    const mutation = useMutation('loginUser', (dto: LoginUserRequestDto) => loginUserRequest(dto));
```

```
    const submitHandler = async (dto: LoginUserRequestDto) => {
        try {
            const response = await mutation.mutateAsync(dto);

            if (response.status !== 200) {
                for (let key in response.errors) {
                    form.setError(response.errors[key].path as FieldPath<LoginUserRequestDto>, {
                        message: response.errors[key].messages[0],
                    });
                }
            } else {
                Cookies.set('token', response.token);
                dispatch(new SetUserAction(mapUserToState(response.user as UserDto)));
            }
        } catch (error) {}
    }

    return <>
        <Typography align={'center'} variant={'h2'}>
            Login to Account
        </Typography>

        <Box m={1} />

        <Typography align={'center'}>
            Please enter your email and password to continue
```

```
            </Typography>

            <Box m={4} />

            <FormProvider {...form}>
              <form autoComplete="off" onSubmit={form.handleSubmit(submitHandler)}>
                <Input name={'email'} label={'Email address:'} />

                <Box m={2} />

                <Input name={'password'} type={'password'} label={'Password:'} />

                <Box m={5} />

                <Box paddingX={2}>
                  <Button type={'submit'} fullWidth variant={'contained'} color={'primary'}>
                    Sign In
                  </Button>
                </Box>
              </form>
            </FormProvider>
          </>
        }

        export const AuthRegisterForm = () => {
```

```
    const { dispatch } = useUserState();

    const form = useForm<RegisterFormValues>({
        resolver: yupResolver(schema)
    });

    const mutation = useMutation('loginUser', (dto: CreateUserRequestDto) =>
createUserRequest(dto));

    const submitHandler = async ({confirmPassword, ...dto}:
RegisterFormValues) => {
        try {
            const response = await mutation.mutateAsync(dto);

            if (response.status !== 200) {
                for (let key in response.errors) {
                    form.setError(response.errors[key].path as
FieldPath<CreateUserRequestDto>, {
                        message: response.errors[key].messages[0],
                    });
                }
            } else {
                Cookies.set('token', response.token);
                dispatch(new SetUserAction(mapUserToState(response.user as
UserDto)));
            }
        } catch (error) {}
    }
```

```
        return <>

            <FormProvider {...form}>
                <Typography align={'center'} variant={'h2'}>
                    Create an Account
                </Typography>

                <Box m={1} />

                <Typography align={'center'}>
                    Create an account to continue
                </Typography>

                <Box m={4} />

                <form                                                    autoComplete="off"
onSubmit={form.handleSubmit(submitHandler)}>
                    <Input name={'email'} label={'Email address:'} />

                    <Box m={2} />

                    <Input name={'name'} label={'Username:'} />

                    <Box m={2} />

                    <Input name={'password'} type={'password'} label={'Password:'}
/>
```

```
                    <Box m={2} />

                    <Input name={'confirmPassword'} type={'password'}
label={'Confirm Password:'} />

                    <Box m={5} />

                    <Box paddingX={2}>
                      <Button type={'submit'} fullWidth variant={'contained'}
color={'primary'}>
                        Sign Up
                      </Button>
                    </Box>
                  </form>
                </FormProvider>
              </>
            }

            export const LighthouseCollectionView = () => {
              const { id } = useParams<{id: string}>();
              const { state } = useProducersState();

              const {
                data,
                isLoading
              } = useQuery(['collection', id], () => getLighthouseCollectionRequest(id));
```

```
    const {
      data: reports,
      isLoading: reportsLoading
    } = useQuery(['reports', data?.collection.id], () =>
getLighthouseReportsRequest({
      take: 100,
      producerId: state.currentProducer?.id as string,
      skip: 0,
      collectionId: data?.collection.id as string
    }))

    if (!data || isLoading) return <></>;

    console.log(data)

    return <>
      <Typography variant={'h1'}>
        Collection: {data.collection.createdAt}
      </Typography>

      <Box m={3} />

      <Grid container spacing={3}>
        <Grid item xs={8}>
          <Paper>
            <Box p={3}>
              <Typography variant={'h3'}>
```

```
                        Reports
                    </Typography>

                    <Box m={2} />

                    <LighthouseReportsList data={reports?.reports ?? []} />
                </Box>
            </Paper>
        </Grid>
        <Grid item xs={4}>
            <LighthouseCollectionCard data={data.collection}
producerData={state.producers.find(i => i.id ==
data.collection.producerId).endpoint} />
        </Grid>
    </Grid>
</>
}

export const LighthouseCollectionCard = ({ data, producerData }: Props) =>
{
    return (
        <Paper>
            <Box p={2}>
                <Typography variant={'h4'}>
                    { data.createdAt }
                </Typography>

                <Typography>
```

```
                { producerData }
              </Typography>

              <Box m={2} />

              <Grid container spacing={1}>
                <Grid item xs={6}>
                  <Box display={'flex'} flexDirection={'column'}
alignItems={'center'}>
                    <ReportScoreProgress
value={data.averagePerformanceScore.desktop} />

                    <Box m={1} />

                    <Typography>
                      <DesktopWindowsOutlinedIcon />
                    </Typography>
                  </Box>
                </Grid>
                <Grid item xs={6}>
                  <Box display={'flex'} flexDirection={'column'}
alignItems={'center'}>
                    <ReportScoreProgress
value={data.averagePerformanceScore.mobile} />

                    <Box m={1} />

                    <Typography>
```

```
                        <PhoneAndroidOutlinedIcon />
                    </Typography>
                </Box>
            </Grid>
        </Grid>

        <Box m={2} />

        <Typography>
            Total Reports: <strong>{data.totalReports}</strong>
        </Typography>
    </Box>
</Paper>
)
}

export const LighthouseDashboard = () => {
    const {state: producersState} = useProducersState();
    const {
        data: reportsResponse,
        isLoading: reportsLoading
    } = useQuery<LighthouseReportsResponseDto>(['reports', producersState.currentProducer?.id], () =>
        getLighthouseReportsRequest({
            skip: 0,
            take: 100,
            producerId: producersState.currentProducer?.id as string
        })
```

```
        );

        const {
            data: reportResponse,
            isLoading: reportLoading
        } = useQuery(
            ['lastReport', producersState.currentProducer?.id],
            () => getLighthouseReportRequest((reportsResponse as LighthouseReportsResponseDto).reports[0].id),
            {
                enabled: !!reportsResponse && reportsResponse.reports.length > 0,
                refetchInterval: false,
                refetchIntervalInBackground: false
            }
        );

        const {
            data: infoResponse,
            isLoading: infoLoading
        } = useQuery(['info', producersState.currentProducer?.id],
            () => getLighthouseInfoRequest(producersState.currentProducer?.id as string)
        );
        const [reports, setReports] = useState(5);
        const changeNumber = (id: string) => {
            setReports(parseInt(id))
            // amount = parseInt(id);
            console.log(id);
```

```
    }

    return !reportsResponse || !reportsResponse.reports.length ?
      (
        <Alert severity={'warning'}>
          You have no generated reports yet. Please, wait for finishing producer jobs.
        </Alert>
      ) :
      (
      <>
        <Grid container spacing={3}>
          <Grid item xs={12}>
            { infoResponse?.info && <LighthouseGeneralInfo data={infoResponse.info} /> }
          </Grid>

          <Grid item xs={12}>
            <LighthouseMinimizedReportInfo
              data={reportResponse?.report as LighthouseReportDto}
              isLoading={reportLoading}
              type={'desktop'}
            />
          </Grid>

          <Grid item xs={12}>
            <LighthouseMinimizedReportInfo
              data={reportResponse?.report as LighthouseReportDto}
```

```
                isLoading={reportLoading}
                type={'mobile'}
              />
            </Grid>

            <Grid item xs={12}>
              <Grid container spacing={3}>
                <Grid item xs={6}>
                  <Paper>
                    <Box p={3}>
                      <Typography variant={'h3'}>
                        Last <DropMenu
                          currentNumber = {reports}
                          recordsLength                                    =
{reportsResponse?.reports.length}
                          onChange={changeNumber}
                        /> reports
                      </Typography>
                      <Box m={2} />

                      <LighthouseReportsList
data={reportsResponse?.reports.slice(0,reports) ?? []} />
                    </Box>
                  </Paper>
                </Grid>

                <Grid item xs={6}>
                  <Paper>
```

```
                        <Box p={3}>
                            <Typography variant={'h3'}>
                                Reports Dynamic
                            </Typography>
                            <Typography variant={'caption'}>
                                Based on the last 10 reports
                            </Typography>

                            <Box m={2} />

                            <LighthouseReportsChart
data={mapReportsToChartData(reportsResponse?.reports.slice(0,reports) ?? [])} />
                        </Box>
                    </Paper>
                </Grid>
            </Grid>
        </Grid>
    </Grid>
</>
    )
}
```

Додаток В

(обов’язковий).

ІЛЮСТРАТИВНА ЧАСТИНА

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ WEB-ДОДАТКІВ. КЛІЄНТСЬКА ЧАСТИНА

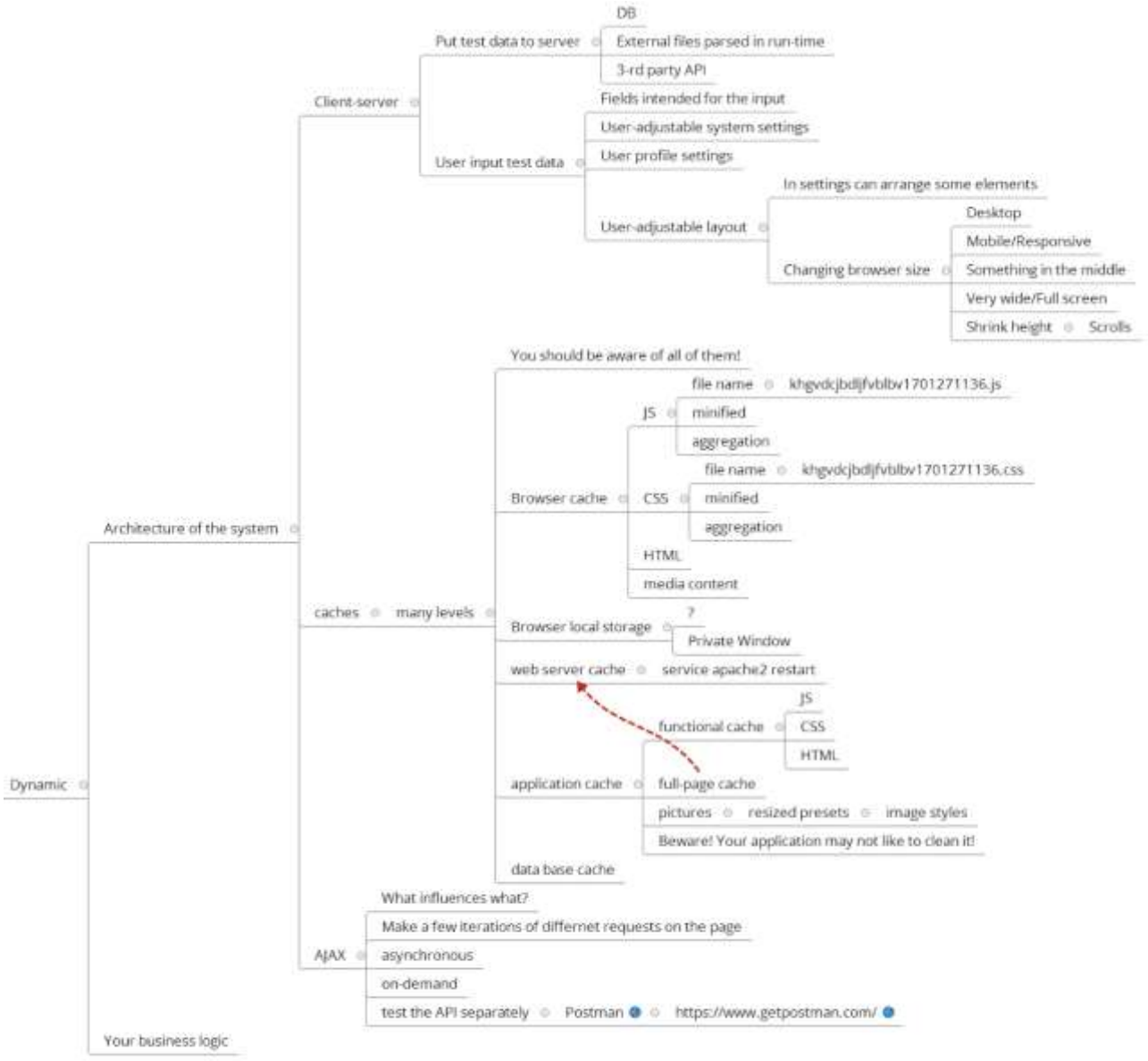


Рис. В.1 – Схема архітектури динамічного тестування

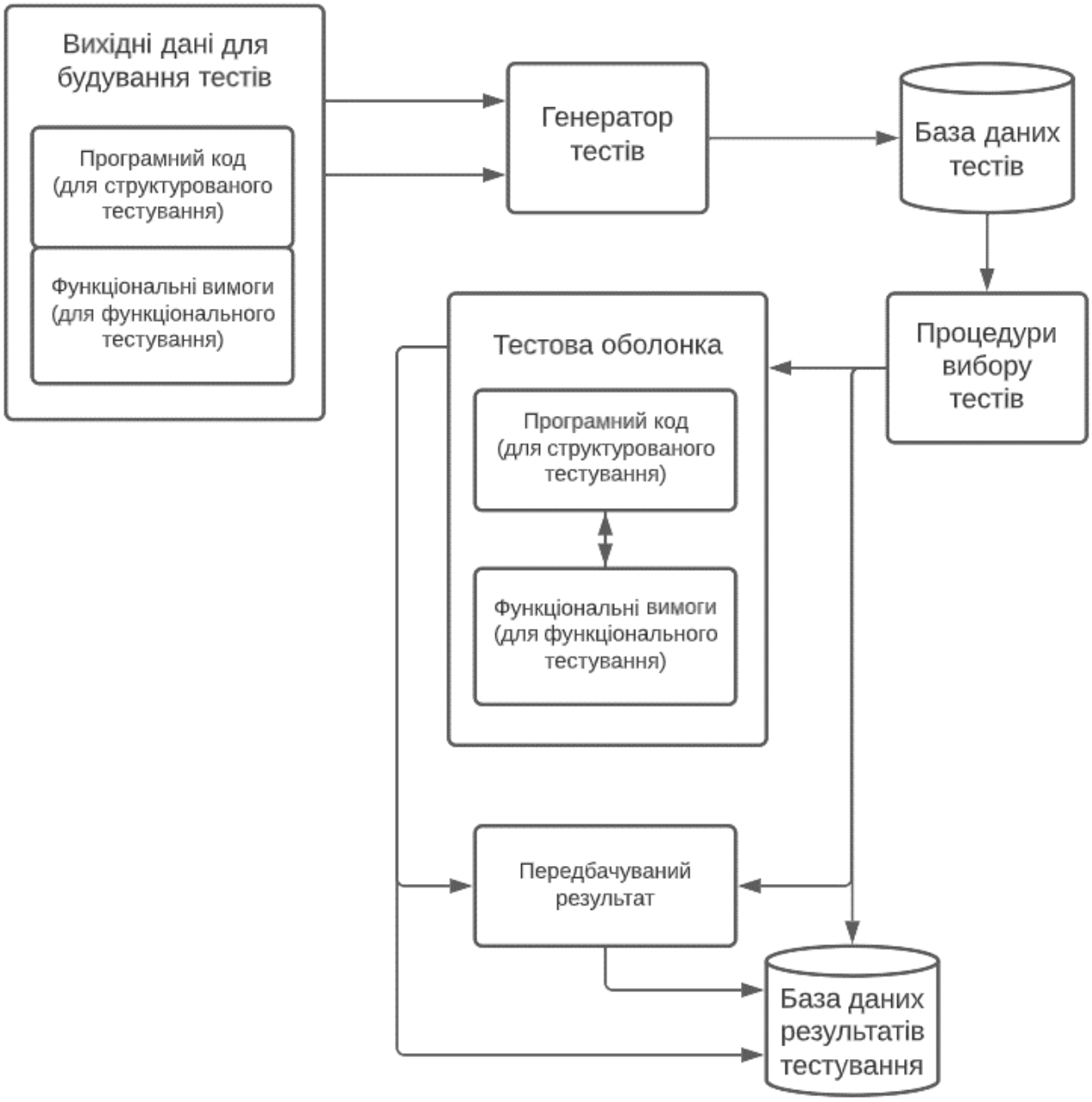


Рис. В.2 – Схема проведення динамічних тестів

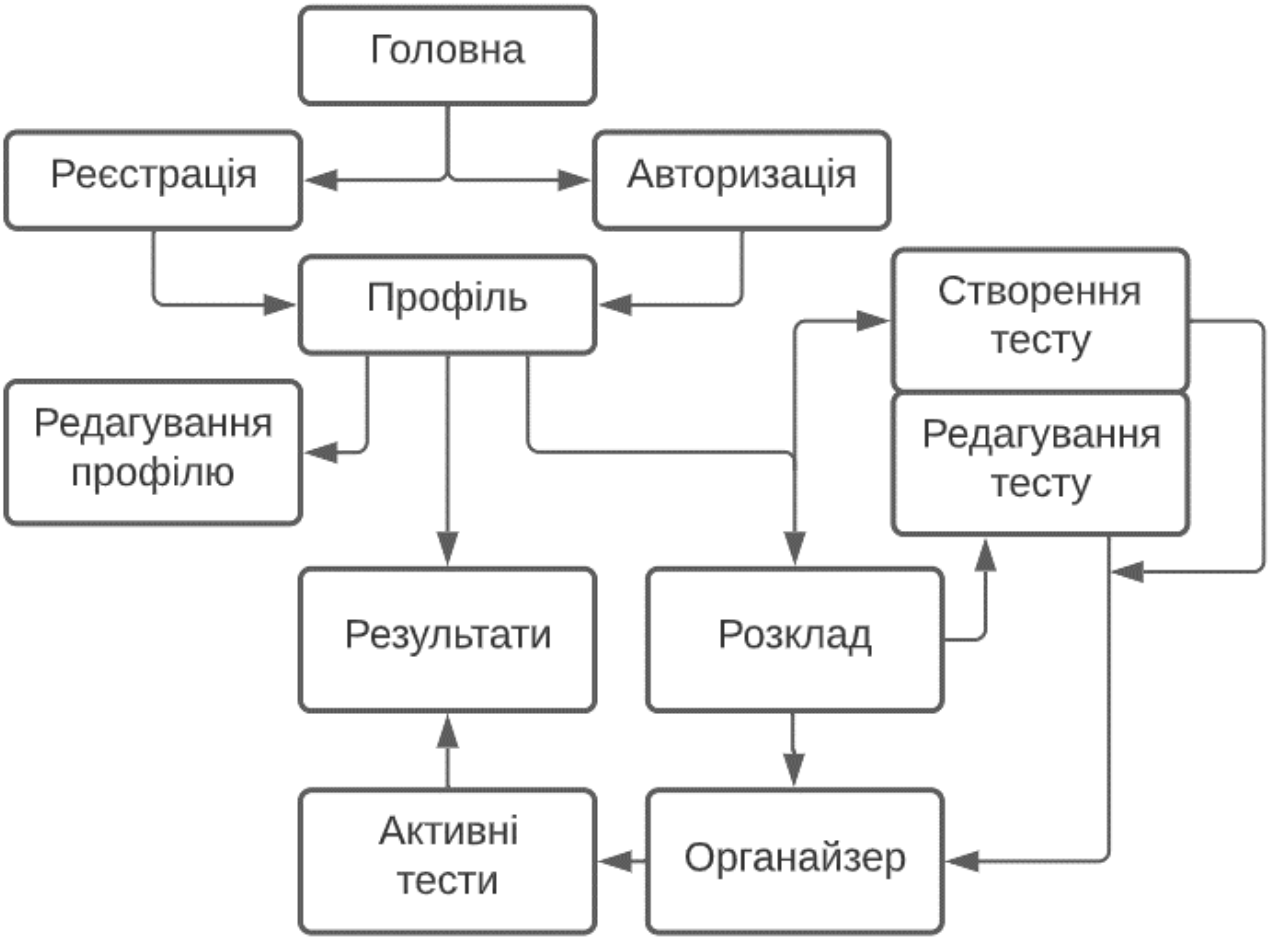


Рис. В.3 – Загальна структурна схема клієнтської частини системи автоматизованого тестування WEB-додатків

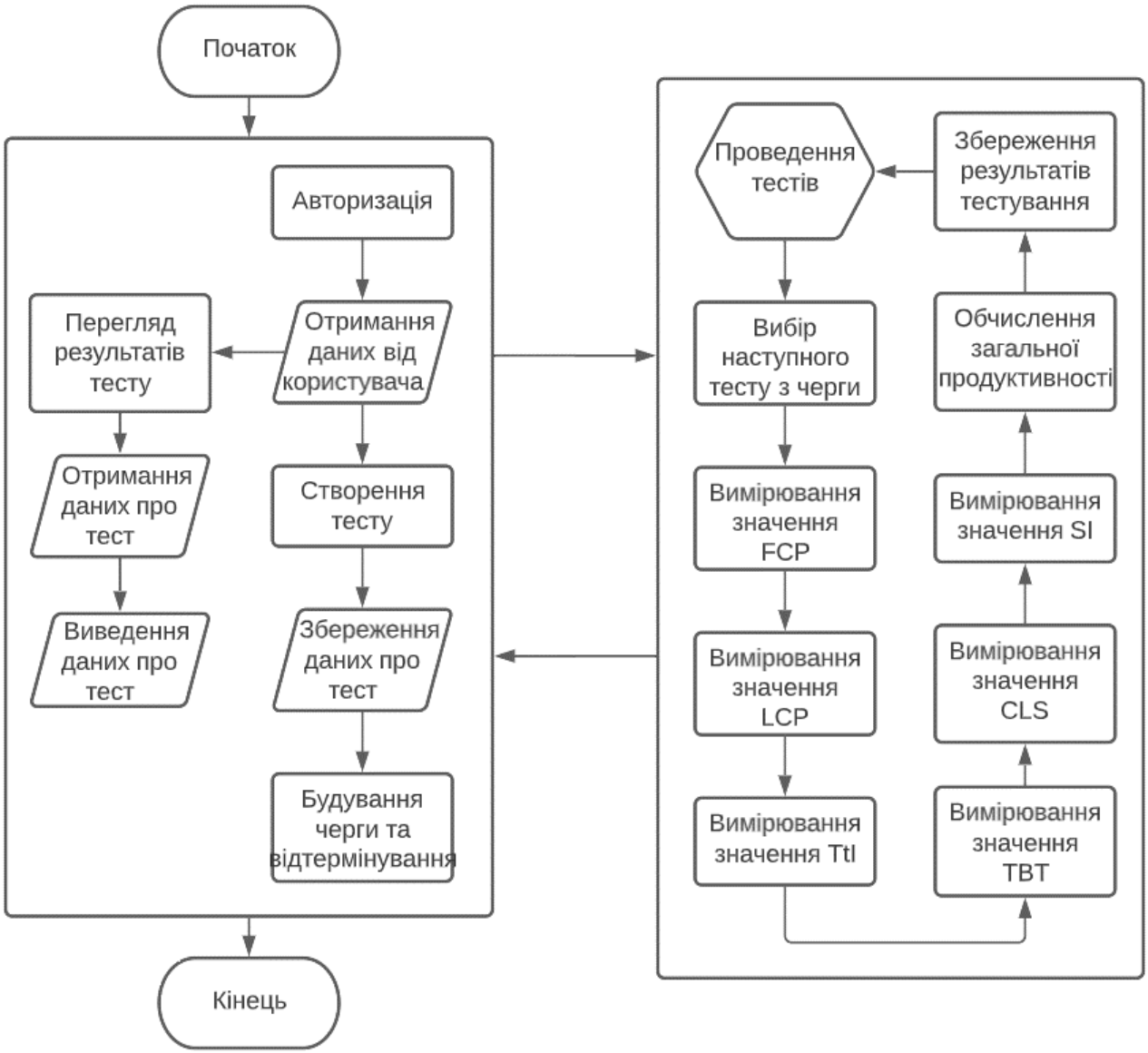


Рис. В.4 – Алгоритм роботи розробленого методу автоматизованого тестування WEB-додатків

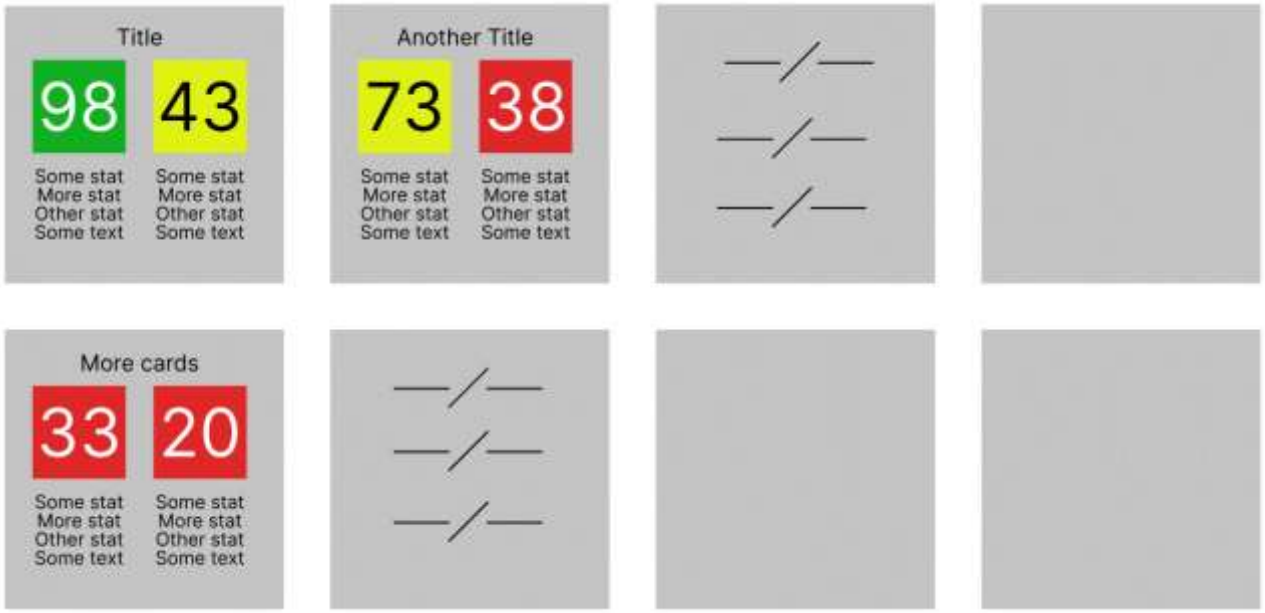


Рис. В.5 – Схема інтерфейсу головної сторінки системи автоматизованого тестування WEB-додатків

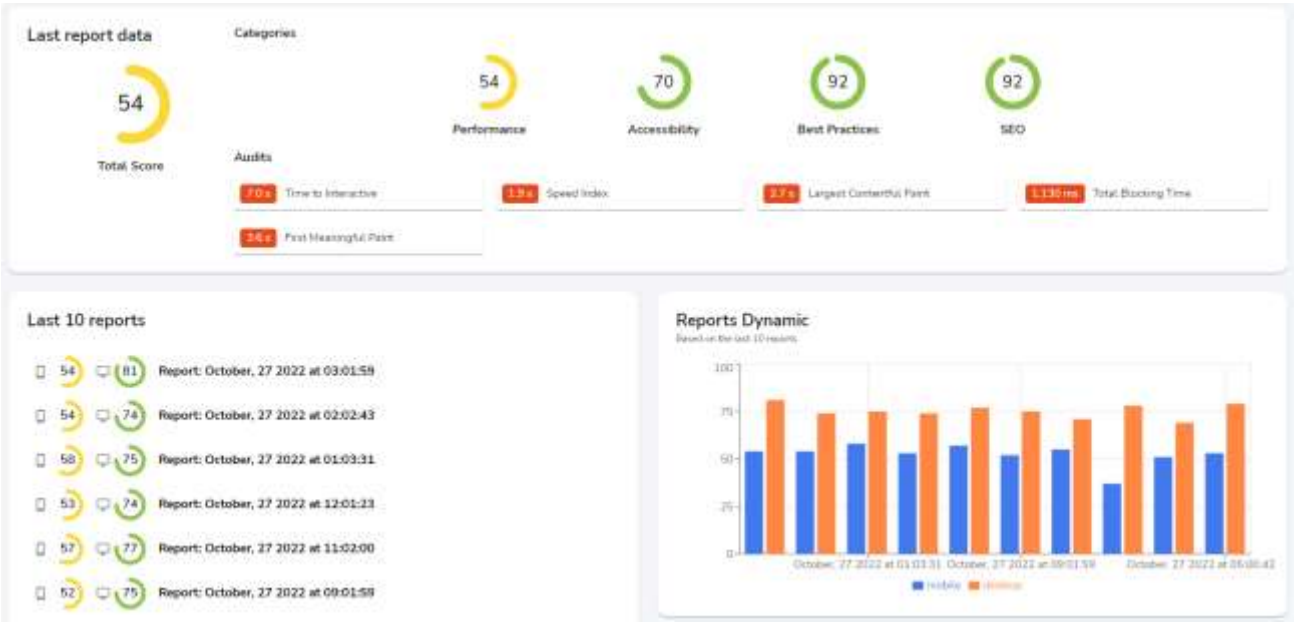


Рис. В.6 – Загальний вигляд головної сторінки програного модуля клієнтської частини автоматизованого тестування WEB-додатків із заповненим розкладом

Рис. В.7 – Загальний вигляд прикладу звіту проведеного тестування у інформаційній технології автоматизованого тестування WEB-додатків

Додаток Г

(обов'язковий).

Інструкція користувача

Для запуску проекту, першим кроком потрібно запустити збудований React-проект у локальній мережі. Відповідно ввівши у командний рядок, у контексті проекту, команду – «npm start».

Після вводу команди термінал виведе повідомлення, що зображене на рисунку Г.1.

```
Starting the development server...
```

Рисунок Г.1 – Повідомлення про початок процесу завантаження проекту у локальній мережі

Через деякий час (5-7 хвилин) проект повноцінно завантажиться, та автоматично буде відкрите посилання у локальній мережі (рис. Г.2).

Рисунок Г.2 – Сторінка авторизації інформаційної технології автоматизованого тестування WEB-додатків

Наступним кроком потрібно зареєструвати користувача у системі, відповідно ввівши дані у форму реєстрації (рис. Г.3).

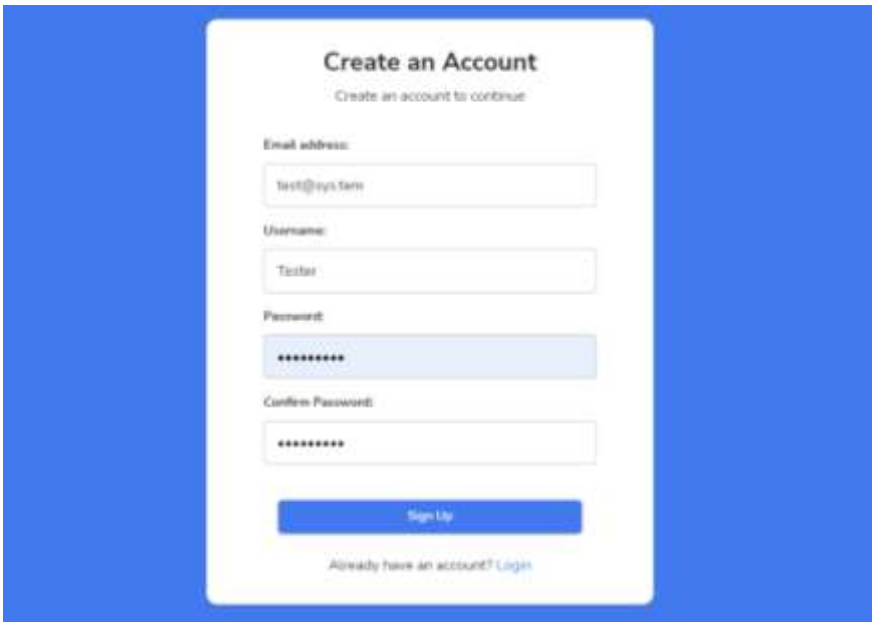


Рисунок Г.3 – Заповнена форма реєстрації користувача інформаційної технології автоматизованого тестування WEB-додатків

Після реєстрації відбудеться перенаправлення на головну сторінку, для нового користувача вона порожня з підказкою, що черга тестів порожня Г.4.

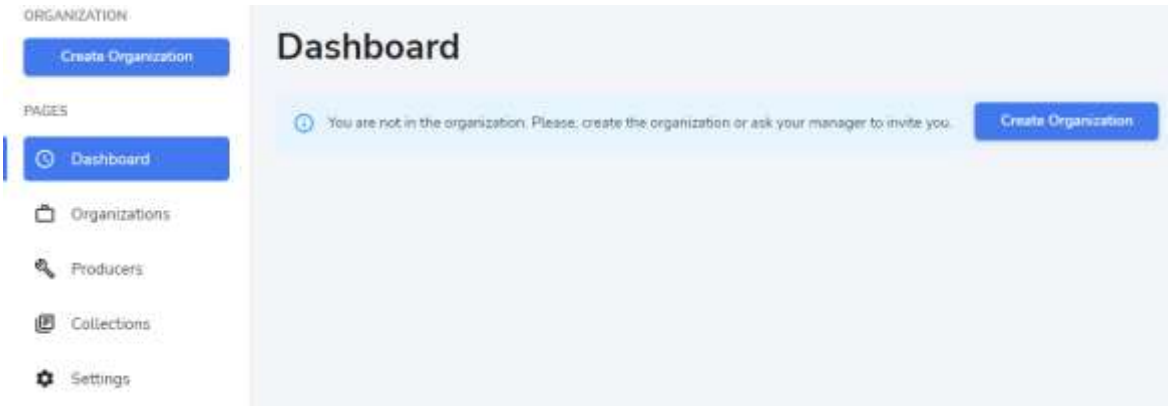


Рисунок Г.4 – Головна сторінка для нового користувача

Натиснувши на кнопку для створення організації, відбудеться перенаправлення на форму її створення (Г.5).

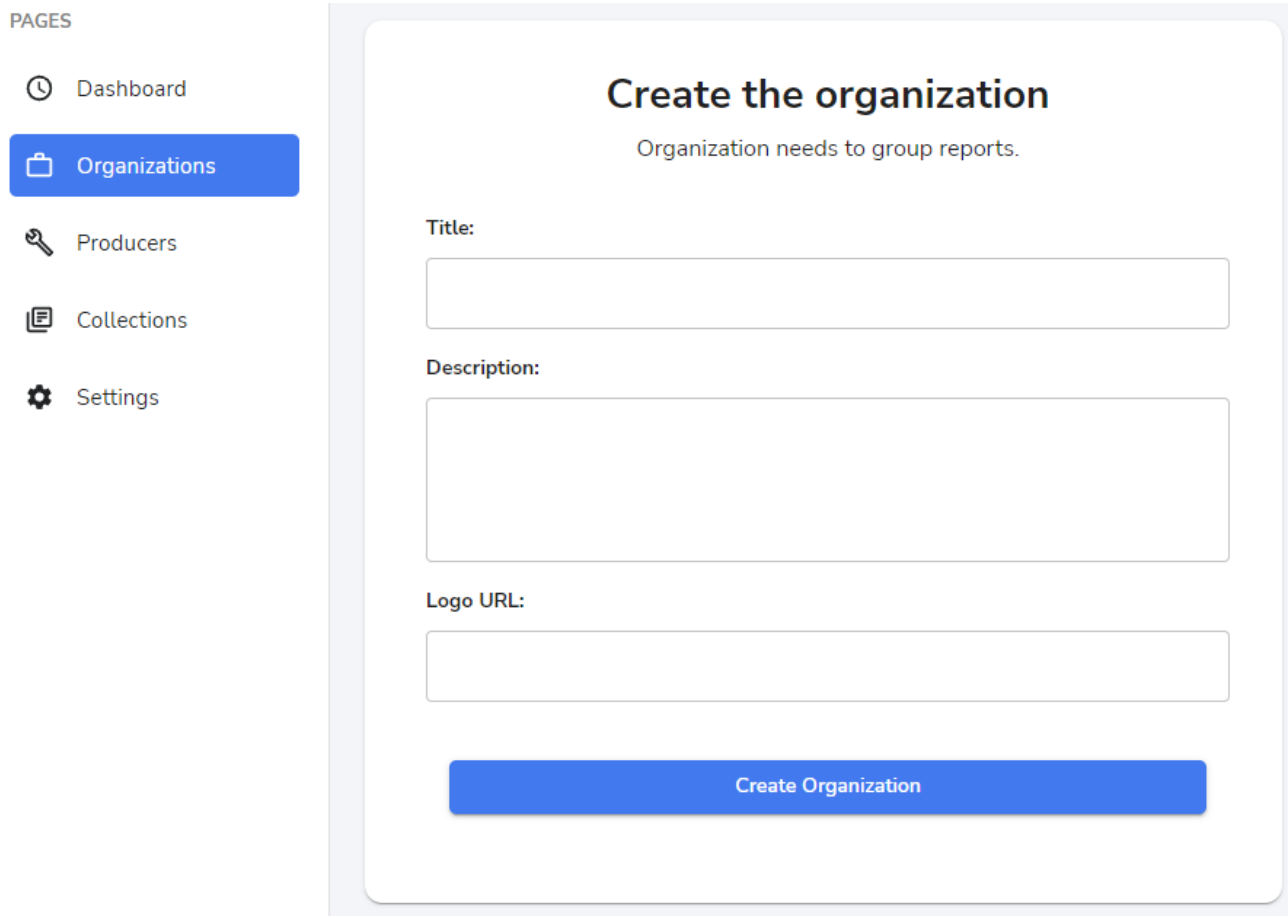


Рисунок Г.5 – Форма створення організації

Для запису відповідного тесту до черги, потрібно заповнити форму на сторінці «Producers» (рис. Г.6).

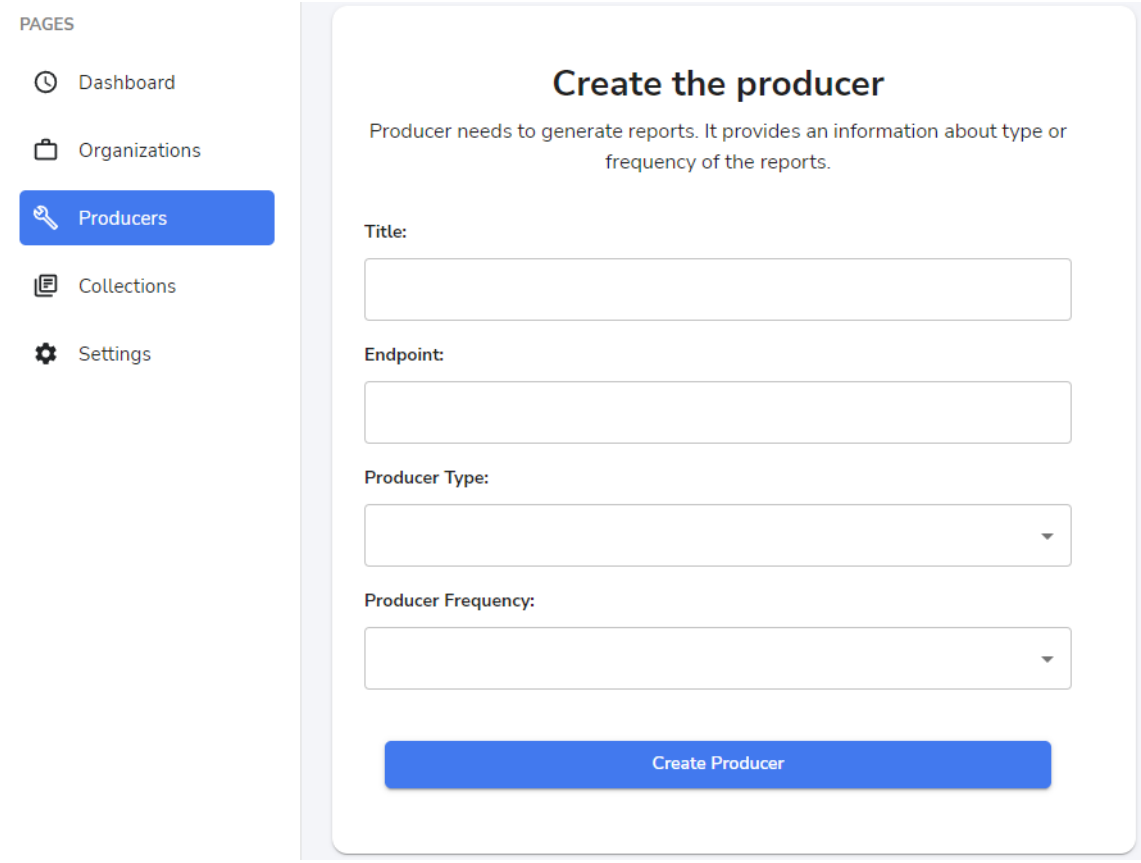


Рисунок Г.6 – Форма запису тесту до черги

Тест записаний у розклад буде відображатись на сторінці «Collections» (рис. Г.7).

Рисунок Г.7 – Загальний вигляд прикладу звіту проведеного тестування у інформаційній технології автоматизованого тестування WEB-додатків

Та після отримання кількох результатів тестування на головній сторінці будуть відображатись останні з них (рис. Г.8).

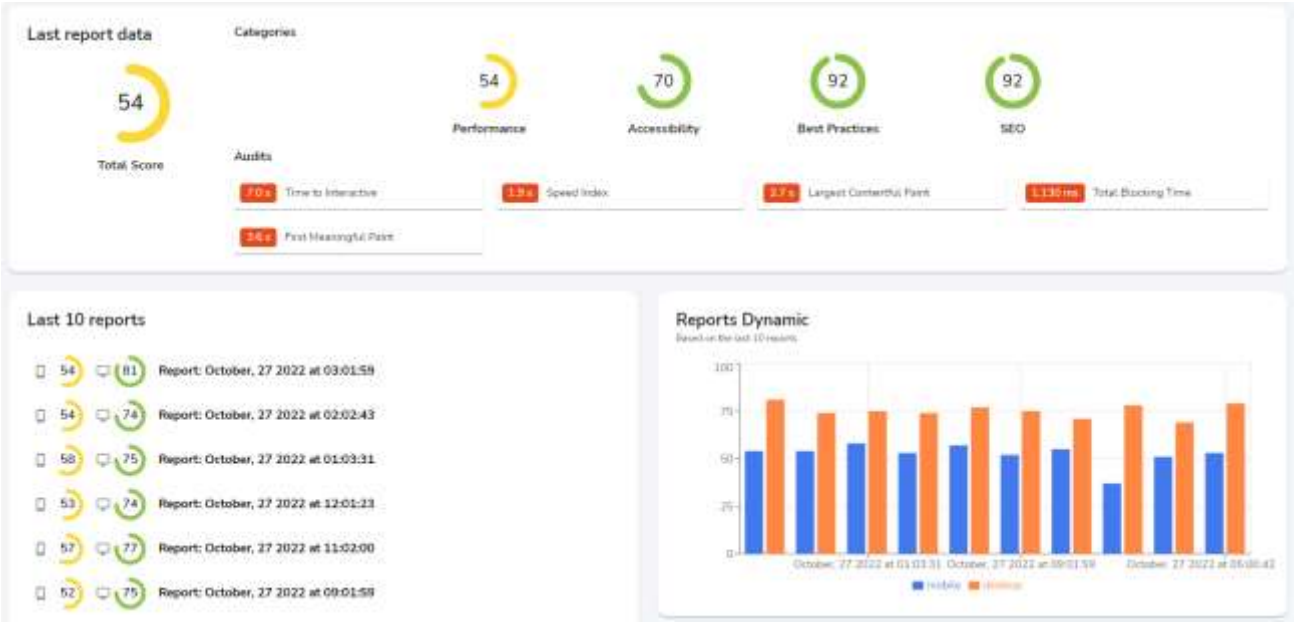


Рисунок Г.8 – Загальний вигляд головної сторінки програного модуля клієнтської частини автоматизованого тестування WEB-додатків із заповненим розкладом