НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

«На правах рукопису»
УДК 004.422.81

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
«____»_____ 2022 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

зі спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Програмне забезпечення для автоматизації тестів»

Виконав:
студент ІІ курсу, групи ІП-12мп
Смоляр Герман Володимирович _____

Керівник:
Старший викладач
Халус Олена Андріївна _____

Рецензент:
доцент кафедри ІСТ, д.т.н., доц.,
Корнага Ярослав Ігорович _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2022 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – другий (магістерський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення інформаційних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Едуард ЖАРІКОВ

«___»_____2022р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Смоляр Герман Володимирович

1. Тема дисертації «Програмне забезпечення для автоматизації тестів», науковий керівник дисертації Халус Олена Андріївна, старший викладач, затверджені наказом по університету від «09» листопада 2022 р. № 4115-с

2. Термін подання студентом дисертації «9» грудня 2022 р.

3. Об’єкт дослідження – програмне забезпечення для автоматизованого тестування

4. Предмет дослідження – інструментальні засоби автоматизованого тестування

5. Перелік завдань, які потрібно розробити – опис предметної області та аналіз існуючих методів; проєктування програмного забезпечення; розроблення програмного забезпечення; дослідження ефективності розробленого програмного забезпечення.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу – ER-діаграма структури бази даних; діаграма використань; діаграма станів програмного забезпечення.

7. Орієнтовний перелік публікацій – тези доповіді на конференції.

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання «1» вересня 2022 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання	Примітка
1	Огляд літератури	1.09.2022	
2	Аналіз існуючих методів автоматизації тестів графічного інтерфейсу	16.09.2022	
3	Проєктування бази даних та серверної частини	2.10.2022	
4	Розробка бази даних	7.10.2022	
5	Розробка нового архітектурного підходу для серверної частини та клієнтської частини	12.10.2022	
6	Розробка діаграм	4.11.2022	
7	Виконання експериментальних досліджень	7.11.2022	
8	Оформлення пояснювальної записки	10.11.2022	
9	Подання дисертації на попередній захист	22.11.2022	
10	Подання дисертації на захист	9.12.2022	

Студент Герман СМОЛЯР

Науковий керівник Олена ХАЛУС

РЕФЕРАТ

Розмір пояснювальної записки — 105 аркушів, містить 7 рисунків, 35 таблиць, 34 джерел, 5 додатків.

Актуальність теми. Тестування відіграє важливу роль на кожному етапі розробки програмного забезпечення, дозволяючи знаходити помилки в програмному коді, на графічному інтерфейсі та відстеження коректності роботи певних послідовних дій. Виділяють дві методики тестування: ручне та автоматизоване тестування. Відмінність між ними полягає в часі виконання та витратних ресурсах. Сутність автоматизації полягає в мінімізації ручного тестування програмного забезпечення. Проте методи впровадження автоматизованого тестування в програмне забезпечення не є зручними та універсальними, що спонукає певні складнощі під час реалізації. Отже, розробка програмного забезпечення, котра надасть можливість добавляти універсальні функції подій автоматизованого тестування є актуальною темою.

Мета дослідження. Основною метою дослідження є спрощення впровадження дій інструментального засобу автоматизованого тестування.

Об’єкт дослідження: програмне забезпечення для автоматизованого тестування.

Предмет дослідження: інструментальні засоби автоматизованого тестування.

Для реалізації поставленої мети **сформульовані наступні завдання:**

– аналіз предметної області;

– огляд існуючих методів та рішень;

– розробити архітектурне рішення, за допомогою якого буде легко впроваджувати дії інструментального засобу автоматизованого тестування без необхідності змін в інших рівнях архітектури;

– забезпечити різновид типів функцій подій;

– розробити вебзастосунок в якому можна керувати тестуванням. А саме: створення тест сценаріїв, створення вхідних даних, створення інформації про проєкт, котрий буде тестуватися.

Наукова новизна результатів магістерської дисертації полягає в тому, що запропоновано архітектурне рішення за допомогою якого впровадження нових дій інструментального засобу автоматизованого тестування відбувається без необхідності змін в інших рівнях архітектури.

Практичне значення отриманих результатів полягає в тому, що розроблена архітектура допомагає легко впроваджувати універсальні функції подій для розробника програмного забезпечення. Кінцевий користувач отримує вебзастосунок в котрому має можливість створювати та запускати тестові сценарії.

Зв'язок з науковими програмами, планами, темами. Робота виконувалась на кафедрі інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

Апробація. Наукові положення дисертації пройшли апробацію на Третій Всеукраїнській науково-практичній конференції молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології» (SoftTech-2022 осінь) – м. Київ.

Публікація. Наукові положення дисертації опубліковано в: Смоляр Г.В. Програмне забезпечення для автоматизації тестів/ Г.В. Смоляр, О.А. Халус // Матеріали Третьої Всеукраїнській науково-практичній конференції молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології» (SoftTech-2022 осінь) – м. Київ: НТУУ «КПІ ім. Ігоря Сікорського», 23-25 листопада 2022 р.

Ключові слова: ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, АРХІТЕКТУРА, АВТОМАТИЗАЦІЯ, ТЕСТУВАННЯ, ТЕСТ СЦЕНАРІЙ.

ABSTRACT

Explanatory note size — 105 pages, contains 7 illustrations, 35 tables, 34 references, 5 applications.

Topicality. Testing plays an important role at every stage of software development, allowing to find errors in the program code, on the graphical interface, and to track the correctness of the work of certain sequential actions. There are two testing methods: manual and automated testing. The difference between them lies in the execution time and expendable resources. The essence of automation is to minimize manual software testing. However, the methods of implementing automated testing in software are not convenient and universal, which leads to certain difficulties during implementation. Therefore, the development of software that will provide the opportunity to add universal functions of automated testing events is a relevant topic.

The aim of the study. The main target is to simplify the implementation of the event functions to the automated testing tool.

The object of research: software for automated testing.

The subject of research: instrumental tools of automated testing.

To archive this goal, the **following tasks** were formulated:

– analysis of the subject area;

– review of existing methods and solutions;

– develop an architectural solution that will make it easy to implement the event function of the automated testing tool without needing changes in other levels of the architecture;

– provide a variety of types of event functions;

– develop a web application in which testing can be managed. It`s creation of test scenarios, creation of data set, creation of information about the project that will be tested.

The scientific novelty of the results of the master`s dissertation is an architectural solution is proposed, with the help of which the implementation of new

actions of the automated testing tool occurs without the need for changes in other levels of the architecture.

The practical value of the obtained results is that the developed architecture helps to easily implement universal event functions for the software developer. The end user receives a web application in which he could create and run test scenarios.

Relationship with working with scientific programs, plans, topics. Work was performed at the Department of Informatics and Software Engineering of the National Technical University of Ukraine «Kyiv Polytechnic Institute. Igor Sikorsky».

Approbation. The scientific provisions of the dissertation were tested at the Third All-Ukranian Scientific and Practical Conference of Young Scientists and Students «Software engineering and advanced information technologies» (SoftTech-2022 autumn) – Kyiv

Publication. The scientific provision of the dissertation was published in: Smoliar H.V. Software for automation testing / H.V. Smoliar, O.A. Khalus // Proceedings of the Third All-Ukranian Scientific and Practical Conference of Young Scientists and Students «Software engineering and advanced information technologies» (SoftTech-2022 autumn) – Kyiv: NTUU «KPI them. Igor Sikorsky», November 23-25, 2022.

Keywords: SOFTWARE, ARCHITECTURE, AUTOMATION, TESTING, TEST SCENARIO.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ІТ — інформаційні технології;

ПЗ — програмне забезпечення;

БД — база даних;

API (application programming interface) — програмний інтерфейс;

REST (representational state transfer) — передача репрезентативного стану;

SQL (structured query language) — мова структурованих запитів.

ВСТУП

Кожного дня у сучасному світі інформаційних технологій з’являється величезна кількість вебсайтів, які розроблені, як продуктове рішення від IT компаній або ж стартап-проєкти. Подальше життя таких проєктів не зупиняється після виходу в інформаційний світ технологій, їх необхідно підтримувати для забезпечення стабільної працездатності, а впровадження нового функціоналу вимагає ретельної перевірки, як сумісність з існуючим функціоналом так й працездатність нового. Для цього інженери програмного забезпечення використовують процес дослідження, котрий називається тестування. Саме цей процес дозволяє виявити на будь-якому етапі розробки проєкту помилки. Адже виявлення помилок на ранній стадії, як наприклад, під час розробки програмного коду, може убезпечити проєкт від фатальної помилки, котра призведе до провалу на ринку. Тож процес розробки проєкту вимагає ретельного тестування на кожному етапі життєвого циклу продукту, що без сумніву вплине на стабільну безпомилкову роботу технічної частини проєкту на ринку.

Для досягнення якісної перевірки функціонала графічного інтерфейсу проєкту, інженери програмного забезпечення використовують два методи тестування: ручне та автоматизоване. Ручне тестування є обов’язковим, оскільки людський фактор перевірки працездатності функціонала підкріплюється. як працюючий, хоч й потребує великого об’єму ресурсів. Тож для мінімізації використання ручного тестування в проєкті використовують автоматизоване тестування, котре симулює дії користувача застосунку та дозволяє запускати тестовий сценарій декілька разів з різними вхідними даними. Для створення автоматизованих тестів розробники звертаються до готових універсальних рішень або до спеціалізованих фреймворків, якщо є необхідність в розробці власного інструментального засобу тестування чи впровадження такого в проєкт. Універсальне рішення — це застосунок, котрий містить основні функції, які необхідні для тестування, такі як: натиснути на кнопку, відкрити нове вікно, перейти за посиланням. Основною перевагою даного рішення є його

використання через графічний інтерфейс користувача, що не потребує знань фреймворку для того, щоб створити тестовий сценарій. Проте проєкт може мати особливість для якого базового функціоналу універсального рішення недостатньо. Тож розробляються застосунки за допомогою фреймворків для автоматизованого тестування або ж впроваджуються в проєкт, котрі будуть відповідати необхідним вимогам.

Фреймворки для автоматизації тестування лише надають готові функції, які розробник може використовувати під час розробки власного інструментального засобу чи створювати за допомогою них тестовий сценарій програмним кодом в проєкті. Написання тестових сценаріїв в проєкті, хоч й швидко реалізується, але не є ефективним рішенням для подальшої довготривалої підтримки, оскільки з часом проєкт набуває нового функціоналу, через, що деякі тестові сценарії необхідно рефакторити, а їх кількість зростає. Тож програмування тестових сценаріїв в проєкті впливає на:

- складність аналізу тесту;
- паралельний запуск тестових сценаріїв;
- запуск тесту в за плановий час;
- розгалуження проєкту.

Розробка власного інструментального рішення займає більше часу для реалізації даного проєкту, проте переваги будуть відчуватися після закінчення розробки. Слід зазначити, що кожний фреймворк надає лиш приклад роботи певної функції, через що, постає питання, як правильно спроєктувати програмний застосунок для автоматизованого тестування графічного інтерфейсу.

Тож метою даної магістерської дисертації є розроблення архітектури для автоматизованого тестування графічного інтерфейсу, що спростить додавання нових функцій в застосунок. За допомогою розробленого архітектурного рішення буде розроблено універсальне рішення для автоматизованого тестування вебзастосунків.

1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Тестування в програмному забезпеченні

Тестування програмного забезпечення — це метод визначення того, чи відповідає фактичний програмний продукт очікуваним вимогам, та перевірки того, що програмний продукт не містить дефектів. Це включає в себе запуск програмного чи системного компонента у своєму темпі з використанням ручних або автоматичних інструментів для оцінки одного або декількох властивостей, програмного забезпечення що цікавлять [1].

Метою тестування програмного забезпечення є виявлення помилок, дефектів або відсутніх вимог, порівняно з фактичними вимогами. Коли проєкт розробки програмного забезпечення перебуває на стадії розробки, необхідно знати, що помилки можуть виникати на будь-якому етапі життєвого циклу ПЗ. Тому важливістю забезпечення якості ПЗ не можна ігнорувати.

Остаточний програмний код, швидше за все, міститиме функціональні та дизайнерські помилки. Тож тестування програмного забезпечення необхідне для виявлення проблем на ранніх стадіях, перш ніж вони будуть викликати проблеми в критичних середовищах. Належно протестовані програмні продукти забезпечують надійність, безпеку та високу продуктивність, що призводить до економії часу, ефективності витрат та задоволеності клієнтів.

Однак слід мати на увазі, що ціна програмного збою може бути дуже високою. Розглянемо основні причини, з яких тестування програмного забезпечення дійсно важливе [2]:

а) допомагає значно заощадити кошти. Економічна ефективність проєкту є однією з основних причин, через яку компанії використовують послуги з тестування, котра складається з багатьох аспектів. Якщо помилки виявляються раніше, витрати на їх виправлення будуть меншими. Тому дуже важливо пройти тестування якнайшвидше перед впровадженням проєкту на ринок. Отже, якщо наймати якісних тестувальників з великим досвідом та технічною підготовкою для свого проєкту — це інвестиції та проєкт буде працездатним;

б) безпека, це ще один важливий момент, який не можна випускати з уваги при тестуванні програмного забезпечення. Вона завжди вважається найуразливішою та найчутливішою частиною ПЗ. У багатьох випадках користувальницька інформація та дані використовуються в комерційних особистісних цілях. Саме тому вважається, що користувачі шукають перевірені та надійні продукти, котрі не містять вразливостей;

в) якість програмного забезпечення важлива. Щоб конкретний продукт працював надійно, він повинен працювати наступним чином:

1) дотримуватися вимог якості у продукті має важливе значення, оскільки це допоможе отримати бажані результати;

2) продукт повинен допомагати користувачеві. Він має забезпечити функціоналом, який пропонує;

3) має бути повністю функціональним, щоб забезпечити ефективну взаємодію з користувачами. Також слід перевірити сумісність пристроїв. Наприклад, під час запуску програми потрібно перевірити всю можливу сумісність з різними операційними системами та пристроями.

г) задоволеність користувачів. Основна мета продукту — забезпечити максимальне задовільнення найприскіпливіших потреб клієнтів. Причина вибору тестування програмного забезпечення пов'язана ще й з тим, що воно забезпечує попередні умови та ідеальну взаємодію з користувачем. Отже, рішення про тестування програмного забезпечення окупиться у довгостроковій перспективі. Здобути довіру клієнтів, безумовно, непросте завдання, але з комплексним підходом є всі шанси на досягнення цієї мети. Саме тому так важливе перше враження, якщо його не справити, користувачі знайдуть інший продукт, який відповідає всім вимогам та задовольнятиме їх потреби на відповідному рівні;

д) управління процесом розробки. Контроль якості допомагає знайти різні сценарії, помилки та відтворити їх. Розробники повинні виправляти помилки, в той час тестувальники програмного забезпечення повинні працювати паралельно

із командою розробників та шукати нові дефекти. Таке поєднання в роботі допомагає прискорити процес розробки;

е) додавання нового функціоналу повинне бути безпроблемне. Програмне забезпечення має завжди адаптуватися до нових функцій, а старі підтримувати внесені зміни. Тому тестування це саме той процес, який перевірить працездатність нового та старого функціоналу;

ж) продуктивність програмного забезпечення. Якщо користувач зіштовхнеться, що ПЗ працює повільно чи має критичні помилки, він буде змушений знайти альтернативу. Оскільки користувач зацікавлений в стабільному, працездатному рішені. Для ІТ-компаній, важливо, щоб користувач завжди залишався задоволений. Тож тестування допомагає протестувати ПЗ на стійкість, а результати нададуть детальну статистику.

Перед тим, як приступати до розробки тестування ПЗ, слід дотримуватися наступних принципів під час тестування ПЗ [3]:

а) тестування відображає наявність дефектів. Ціль тестування ПЗ полягає в тому, щоб виявити наявні дефекти в проєкті, тож чим частіше проводити тестування ПЗ, тим зменшується кількість дефектів. Варто зазначити, тестування визначає тільки наявність дефекту в проєкті, проте не гарантує його відсутність, навіть, якщо проводити регулярне тестування ПЗ;

б) досконалого тестування не існує. Досконале тестування — це тестування функціонала проєкту в усіх ймовірних вхідних даних, таких як: дійсних, недійсних, попередніх умов. Реалізація такого тестування неможлива, оскільки ПЗ не зможе протестувати кожний тестовий сценарій з усіма даними. Причиною неможливості є недоцільна витрата великого об’єму ресурсів проєкту, оскільки для цього необхідно більше витрат на тестування, зусиль та часу;

в) попереднє тестування. Тестування необхідно проводити на всіх етапах розробки ПЗ. Виявлення дефектів в ПЗ на ранній стадії розробки, коштуватиме проєкту набагато дешевше, ніж цей дефект буде виявлено після його

впровадження на ринок. Такий підхід дозволить уникнути дефектів на ранній стадії розробки;

г) кластеризація дефектів. ПЗ може містити велику кількість модулів, через що, слід розподілити ресурс тестування пропорційно відносно наявності кількості дефектів в модулі ПЗ;

д) парадокс пестициду. Тестування сценарію з однаковими даними не гарантує відсутність дефекту, оскільки для одного типу вхідних даних тестовий сценарій виконується успішно, то для інших може містити дефект у виконанні. Слід виконувати тестовий сценарій з різними вхідними даними для виявлення нових дефектів;

е) тестування є залежним від його середовища. Підхід до тестування може відрізнятися в залежності від його середовища. Наприклад, тестування вебзастосунку відрізняється від тестування ПЗ на базі iOS. Тож кожне середовище має свій підхід для виконання тестування;

ж) дефект завжди присутній. Якщо тестування не відображає наявність помилок чи дефектів в ПЗ, це не є фактором готовності ПЗ. Програмне забезпечення повинне відповідати всім поставленим вимогам користувача.

Також під час розроблення тестів для програмного забезпечення слід дотримуватися наступних правил:

а) тестування згідно ролі програміста на проєкті. Тестування ПЗ командою розробників недопустиме, оскільки після довготривалої розробки ПЗ, розробник проводить тестування не об’єктивно. Тож тестування виконується командою тестувальників, які мають деструктивний підхід до проєкту. Проте команда розробників може проводити модульне та інтеграційне тестування;

б) програмне забезпечення не може бути ідеально працездатним. Ніколи не можна гарантувати, що в ПЗ відсутні помилки чи дефекти, адже це неможливо підтвердити;

в) тестування на кожному етапі розробки. Тестування починається з першого етапу розробки ПЗ, а саме з аналізу вимог. Дотримання цього підходу дозволить уникнути розширення ланцюгу дефектів;

г) обмежений час. Тестування повинне займати визначений час для реалізації ефективного плану тестування. Цей план необхідно планувати заздалегідь та визначити критерії згідно з яких припиняється процес тестування;

д) різні вхідні дані. Для відтворення всіх ймовірних випадків тестування необхідно використовувати різні вхідні дані такі, як: правильні, тестові, негативні та неочікувані;

е) аналіз виконаних тестів. Правильна перевірка результатів тесту матиме наступний вигляд: необхідно провести кількісну оцінку тесту та його результатів. Під час перевірки результатів тестових випадків на документацію слід посилатися відповідним чином, щоб забезпечити належне тестування. Тестування має максимально підтримуватися автоматизованими інструментами та методиками. Крім того, щоб переконатися, що система робить те, що вона має робити, тестер повинен переконатися, що система не робить те, що вона не повинна робити відповідно;

ж) перевірка гіпотез. Тестові приклади ніколи не варто розробляти на основі гіпотез або припущень. Їх завжди слід належним чином перевіряти. Зокрема, якщо припустити, що програмний продукт розроблено без будь-яких помилок у розробці тестів, це може призвести до істотно слабких результатів тестів.

1.2 Типи тестування

Статичне тестування — це тестування програмного забезпечення, в якому не використовується реальне програмне забезпечення або додаток. Цей метод тестування вимагає, щоб розробники читали код та знаходили помилки вручну [4]. Такий тест програм не вимагає окремих додатків для перегляду, проходження та перевірки. Особливості статичного тестування:

- тест перевіряє код, вимоги та проєктну документацію;
- покращує стандарти програмного забезпечення, шукаючи помилки, недоліки коду та потенційно шкідливий код у ПЗ;

– статичне тестування починається на початку життєвого циклу програмного забезпечення і називається перевірним тестуванням;

– статичні тести виконуються на ранній стадії процесу розробки до фактичного запуску коду. Це відрізняється від динамічних тестів, що запускаються після виконання коду;

– статичне тестування можна виконувати за допомогою таких документів, як специфікації вимог, проєктна документація, вихідний код, плани тестування, тестові сценарії, тестові набори або вміст вебсторінки.

Динамічне тестування — це тип тестування програмного забезпечення, при якому системний код компілюється і виконує дії в системі запущеного середовища. Це метод оцінки здійсненності програмного забезпечення шляхом надання вхідних даних та вивчення вихідних даних. Основною метою даного тестування є забезпечення правильної роботи та стабільності без суттєвих дефектів під час та після встановлення програмного забезпечення [5].

Особливості динамічного тестування:

– перевірка є частиною процесу верифікації та підтвердження;

– складніше, ніж традиційне тестування програмного забезпечення. Оскільки тестувальнику необхідно знати систему, а також розуміння того, як система реагує на дані, що вводяться;

– кращі та більш реалістичні результати, ніж в статичному тестуванні;

– допомагає знайти помилки, які не можуть бути виявлені при звичайному тестуванні програмного забезпечення;

– використовується для регресійного тестування.

В таблиці 1.1 визначені відмінності між статичним та динамічним тестуванням по критеріях.

Таблиця 1.1 — Відмінності між статичним та динамічним тестуванням

Критерій	Статичне тестування	Динамічне тестування
Виконання тесту	Виконується на ранніх стадіях розробки ПЗ	Виконується на пізніх стадіях розробки ПЗ
Проведення тесту	Тест проводиться без запуску програми	Тест проводиться шляхом запуску певної програми
Альтернативна назва типу тестування	Верифікаційний тест	Тест підтвердження
Зміст тесту	Складається з оглядів, покрокових інструкцій, перевірок тощо	Складається з функціонального, нефункціонального тестування та аналізу потоку даних/управління
Робота, котра виконується	Оцінює код та документацію	Знаходить помилки та слабких місцях ПЗ
Як запускається тест	Запускає сухий прогін коду як частину статичного аналізу коду	Код повністю проаналізовано різними способами виконання
Час виконання	Зазвичай не займає багато часу	Так, як це включає кілька тестів, це зазвичай займає багато часу
Що охоплює під час виконання	Охоплює структурні тести та тести покриття тверджень	Охоплює виконуваний код
Що включає до себе	Включає документи з вимогами, проєктну документацію, специфікації додатків	Включає модульне тестування, тестування інтеграції, тестування системи, тестування продуктивності, тестування безпеки тощо

1.2.1 Функціональне тестування

Функціональне тестування гарантує, що робота додатків або мобільної програми виконується відповідно до технічних та бізнес вимог. Функціональні тести запускаються вручну перед нефункціональними тестами [6]. Тестувальник надає програмі певні вхідні дані та порівнює дані з очікуваним результатом.

Розглянемо типи функціонального тестування:

а) димовий тест (Smoke test), використовується перед фактичним тестом для переконання, що основні функції працюють правильно перед налаштуванням інших етапів тесту;

б) тест на узгодженість (Sanity test), після виправлення помилки або добавлення нового функціоналу необхідно впевнитися, що ця зміна не викликає нових помилок;

в) модульний тест (Unit test), використовується для перевірки окремих блоків чи компонентів системи;

г) інтеграційні тести (Integration test), запускаються для перевірки зв'язку між різними компонентами системи. Необхідні для переконання, що різні частини системи працюють разом належним чином. Тест інтеграції можна запускати вручну або автоматично;

д) перевірка прикордонних значень (Boundary value test) – це метод тестування чорної скриньки, який включає тестування програмного забезпечення з вхідними значеннями, що знаходяться на межі певного діапазону допустимих значень;

е) API тест (API test) — це тип тесту, який фокусується на перевірці функціональності взаємодії інтерфейсу програми. Тести API зазвичай виконуються за допомогою автоматизованих інструментів, але інколи використовується ручне тестування.

ж) приймальне тестування (User Acceptance Test), перевіряє чи програмне забезпечення відповідає бізнес-вимогам користувачів. Зазвичай виконується групою користувачів, які представляють передбачувану аудиторію;

з) регресійний тест (Regression test) — це тестування програмного забезпечення, яке перевіряє, чи програмна система відповідає вимогам навіть після внесення змін. Ціль та мета регресійного тестування переконатись, що зміни не приносять до програми нових помилок;

и) тестування сумісності (Interoperability test) гарантує, що програма може взаємодіяти з іншими програмними системами та компонентами без проблем;

к) тестування інтерфейсу (GUI test) — перевіряє графічний інтерфейс користувача програмного забезпечення. Мета полягає в тому, щоб впевнитися, що функціонал ПЗ працює коректно згідно його графічного інтерфейсу.

1.2.2 Нефункціональне тестування

Нефункціональне тестування досліджує усі аспекти, які не охоплені функціональним тестуванням. Охоплює продуктивність, доступність, масштабованість та надійність програмного забезпечення. Запуск таких тестів необхідний для переконання, що кінцевий користувач програми буде задоволений. Для ІТ-компаній важливо, щоб продукт відповідав очікуванням користувача, саме тоді продукт стає конкурентоспроможним на ринку. Даний тип тестування потребує від тестувальника більшої винахідливості та креативу. Тестувальники повинні розробити стратегії для виявлення очікувань клієнтів та надати набір тестів, щоб гарантувати, що ці очікування виправдались. Результати нефункціонального тесту вимірюються за шкалою, у той час, як функціональні тести визначають, як працює програма [7].

Розглянемо типи нефункціонального тестування:

а) тестування доступності (Availability test) використовується, щоб переконатися, що система доступна, коли вона потрібна користувачам;

б) тест продуктивності (Performance test) перевіряє продуктивність та ефективність програмних систем;

в) тест на сумісність(Compatibility test). Цей тип тестування перевіряє, наскільки добре продукт взаємодіє з іншими компонентами, такими як операційні системи, браузери та різноманітне обладнання;

г) тестування локалізації (Localization test) гарантує, що продукт відповідає вимогам місцевої аудиторії;

д) об'ємний тест (Volume test) перевіряє, наскільки ефективно працює система за умов підвищеного обсягу оброблюваних даних;

е) тестування масштабованості (Scalability test) — це методологія тестування програмного забезпечення, яка гарантує, що продукт може бути розширений для задоволення потреб кінцевого користувача;

ж) тестування зручності (Usability test), використовується для перевірки зручності використання програми для клієнтів;

з) перевірка надійності (Reliability test), спосіб тестування програмного забезпечення, котрий поєднує в собі стрес-тестування, тестування безпеки та функціональне тестування, щоб визначити, чи програмний продукт відповідає функціональним критеріям;

и) тест безпеки (Security test), необхідний для виявлення всіх можливих недоліків програми та визначити, наскільки добре захищені конфіденційні дані клієнтів та внутрішні ресурси застосунку;

к) навантажувальне тестування (Load test) — це знання того, як програма реагує на певне навантаження. Необхідне для забезпечення стабільної продуктивності для користувачів. Це дозволяє оцінити продуктивність системи при високих пікових навантаженнях;

л) випробування витривалості (Endurance test) — форма тестування, яка використовується з метою оцінки ефективності програми при великому навантаженні. Порівняно з навантажувальним тестування, навантаження постійно збільшується і зберігається;

м) тест на відповідність (Compliance test) — метод тестування програмного забезпечення, який гарантує, що продукти відповідають

глобальним стандартам розробки програмного забезпечення, які часто розробляють багатонаціональні корпорації;

н) стрес тестування (Stress test), виконується для того, щоб побачити, як саме програмне забезпечення працює при різних навантаженнях та навантаженнях на функціональність програми;

о) тестування переносимості (Portability test) визначає, наскільки легко перенести програмний компонент або програму з одного обладнання чи операційної системи на інше;

п) тест аварійного відновлення (Disaster recovery test) використовується для оцінки того, скільки часу займає відновлення і як програма відновлює дані після збою або збою мережі.

1.2.3 Тестування чорної скриньки

Тестування, коли невідомо функціонал ПЗ або ж відсутні внутрішні знання проєкту називають тестування чорної скриньки [8].

Існують наступні способи, за допомогою яких можна провести тестування чорної скриньки:

а) тестування синтаксисом, застосовується до систем, які можуть синтаксично бути представлені певною існуючою мовою. Наприклад, це є компілятор. Головною особливістю даного способу є генерація тесту, котре містить кожне граматичне правило;

б) поділ еквівалентності є другим типом тестування. Нерідко можна побачити, як багато типів вхідних даних працюють ідентично, тому можна згрупувати їх і перевіряти лише один вхідний сигнал на групу, а не подавати їх окремо. Головний задум полягає в тому, щоб розділити вхідну область системи на кілька класів еквівалентності таким чином, щоб кожен член класу працював однаково, тобто якщо тестовий приклад в одному класі викликає деякі помилки, інші члени класу також матимуть той самий результат і призведуть до тої ж помилки. Ця техніка включає два етапи:

1) ідентифікація класу еквівалентності. Розділення поля введення, як мінімум на два набори, припустимі та неприпустимі значення. Наприклад, якщо допустимий діапазон становить від 0 до 100, виберемо одне допустиме введення, наприклад 14, і одне недопустиме введення, наприклад 111;

2) генерація тестових випадків. Перший спосіб генерації, кожному допустимому та неприпустимому вхідному класу надається унікальний ідентифікаційний номер. Другий спосіб, створення тестових наборів, які охоплюють всі допустимі та недійсні тестові набори, припускаючи, що жодні два неприпустимі, введення не маскують один одного. При обчисленні квадратного кореня числа класу еквівалентності маємо наступні прийнятні вихідні дані:

– ідеальне квадратне число виводить ціле число;

– цілі числа, які є повними квадратами, виводять десяткові числа;

– позитивне десяткове число;

– негативне число (ціле або десяткове);

– нечислові символи, такі як 'a', ', !', ', ;'.

в) аналіз кордонів. Кордони є областями, схильні до помилок. Отже, якщо тестові приклади розроблені з урахуванням граничних значень вхідної області, ефективність тестування підвищується та збільшується ймовірність виявлення помилок. Наприклад, якщо допустимий діапазон становить від 10 до 100, необхідно перевірити наявність 10 та 100 на добавлення до допустимих та неприпустимих введень;

г) причинно-наслідкові діаграми. Цей метод встановлює відносини між логічними вхідними даними, званими причинами, та відповідними діями, званими наслідками. Алгоритм визначення причин та наслідків представлень реалізується за допомогою булевих графіків:

1) визначити входи (причини) та виходи (наслідки);

2) створити діаграми причин та наслідків;

3) перетворити діаграму на таблицю рішень;

4) перетворити правила таблиці рішень на тестові приклади.

Розглянемо приклад роботи алгоритму на рисунку 1.1, позначивши «П» — причина, «Н» — наслідок.

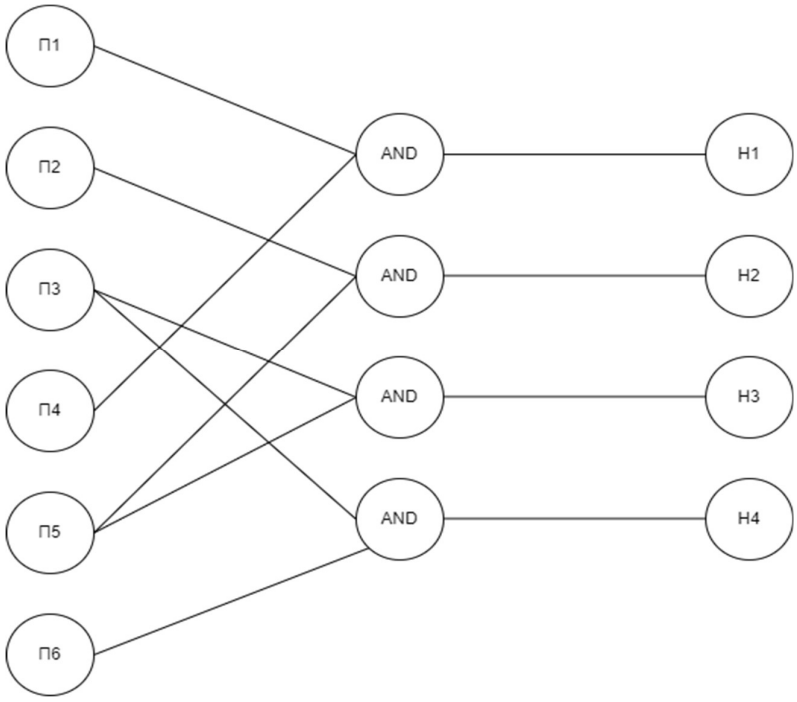


Рисунок 1.1 — Причинно-наслідковий граф

Цей граф можна конвертувати в таблицю рішень, приклад представлений в таблиці 1.2.

Таблиця 1.2 — Рішення для причинно-наслідкового графа

Логічні вхідні дані	Позначення	Стовпець відношення №1	Стовпець відношення №2	Стовпець відношення №3	Стовпець відношення №4
Причинна	П1	1	0	0	0
	П2	0	1	0	0
	П3	0	0	1	1

Продовження таблиці 1.2

Причина	П4	1	0	0	0
	П5	0	1	1	0
	П6	0	0	0	1
Наслідок	Н1	X	-	-	-
	Н2	-	X	-	-
	Н3	-	-	X	-
	Н4	-	-	-	X

Кожен стовпець таблиці відповідає правилу, яке стане тестовим прикладом для тестування, тобто отримали 4 тести.

д) тестування на основі вимог — включає перевірку вимог, зазначених у програмній системі SRS;

е) перевірка сумісності. Результати тестування залежать не тільки від продукту, а й інфраструктури забезпечення функціональності. Очікується, що продукт працюватиме правильно після зміни параметрів інфраструктури. Деякі установки, які зазвичай впливають на сумісність програмного забезпечення:

– процесор;

– архітектура ПЗ та характеристики комп’ютера;

– віддалені сервери, наприклад база даних розташована на хмарному середовищі;

– операційна система.

Серед переваг тестування методом чорної скриньки відносять:

а) тестувальникам не потрібно мати додаткові функціональні знання або ж навички програмування для реалізації тестів чорної скриньки;

б) доцільно використовувати під час реалізації тестів на великих системах;

в) тестування проводиться з погляду користувача чи клієнта;

г) тестові випадки легко відтворити;

д) використовується для виявлення неоднозначностей та невідповідностей у функціональних специфікаціях.

Недоліками тестування методом чорної скриньки є:

а) одні й самі тести можуть бути повторні під час реалізації процесу тестування;

б) відсутність чіткої функціональної специфікації, тестові приклади важко реалізувати;

в) тестові випадки складні у виконанні через складність вхідних даних на різних етапах тестування;

г) може бути неможливо визначити причину збою тесту;

д) не виявляє помилок в адміністративних структурах;

е) обробка великих вибірок вхідних даних може займати виділені ресурси та займати багато часу.

1.2.4 Тестування білої скриньки

Методи тестування білої скриньки, як і тестування чорної скриньки, аналізують не тільки функціональність, а й внутрішнє групування, структури даних, що входять у внутрішню структуру, структуру коду й роботу програмного забезпечення [9]. Метод білої скриньки також називають тестом скляної скриньки, тестом прозорої скриньки або структурним тестом.

Робочий процес тестування білої скриньки має наступний вигляд:

а) вхідні дані: вимоги, функціональні характеристики, проєктна документація, вихідний код;

б) обробка: проведення аналізу ризиків підтримки всього процесу;

в) правильне планування тестування: розробка тестових сценарії, що охоплюють код. Повторення тестування, доки не буде досягнуто безпомилкового програмного забезпечення. Також повідомлення про результати;

г) результат: підсумковий звіт з усього процесу тестування.

Розглянемо, які існують техніки для тестування білою скринькою:

а) покриття оператора. Мета цього методу пройти весь оператор хоча б один раз. Таким чином перевіряється кожен рядок коду. Для блок-схем кожен вузол має бути пройдений принаймні одноразово. Кожен рядок коду покритий, що допомагає виявити проблемний код.

б) покриття гілок. У цьому методі тестові приклади розроблені таким чином, що кожна гілка кожної точки прийняття рішення проходить щонайменше один раз. Блок-діаграма має проходити через кожне ребро хоча б одноразово. На рисунку 1.2 зображено алгоритм роботи покриття оператора;

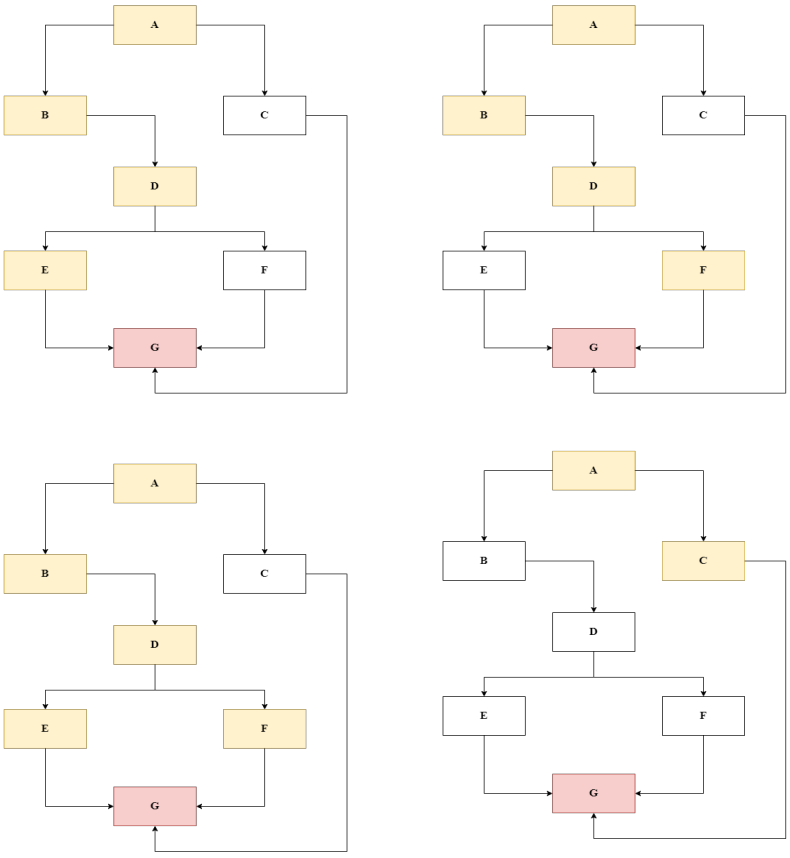


Рисунок 1.2 — Метод покриття гілок

в) покриття умов. Всі індивідуальні умови мають бути покрити, для прикладу опишемо такий випадок:

1) прочитати вхідні дані, X та Y;

2) перевірити чи X дорівнює 0 або ж Y дорівнює 0;

3) якщо умова виконується, звітуємо про виконання.

В цьому прикладі є перевірка двох умов, X дорівнює 0 та Y дорівнює 0. Результатом цього тесту буде вірно або невірно в залежності від їх значень. Успішне виконання даного тесту, якщо X дорівнює 2 та Y дорівнює 0 або ж навпаки;

г) Покриття кількох умов. Всі можливі комбінації перевіряються принаймні один раз. Наприклад:

1) X дорівнює 0 та Y дорівнює 0;
2) X дорівнює 0 та Y дорівнює 2;
3) X дорівнює 22 та Y дорівнює 0;
4) X дорівнює 22 та Y дорівнює 2.

В підсумку маємо 4 тести для 2 умов, а отже отримуємо, що для n-кількості тестів буде потрібно 2n-кількості тестових випадків;

д) базове тестування шляху. За допомогою цього методу будується граф потоку керування з коду або ж блок-схеми. Цей граф обчислює циклометричну складність визначення наявних незалежних шляхів, в результаті створює мінімальну кількість тестових випадків для кожного незалежного шляху. Алгоритм створення базового тестування шляху:

1) створити блок-схему керування;
2) розрахувати циклометричну складність;
3) знайти незалежний шлях;
4) розробити тестові приклади для кожного незалежного шляху.

е) позначення потокового графа. Це спрямований граф, що складається з вузлів та ребер. Кожен вузол є серією тверджень або точок прийняття рішень. Вузол-предикат — це вузол, що представляє точку прийняття рішення, що містить умови, за яких граф розбивається. Слід зазначити, що області обмежені вузлами та ребрами. На рисунку 1.3 зображено потокові графи для наступних операторів в програмуванні: if-else, until, while.

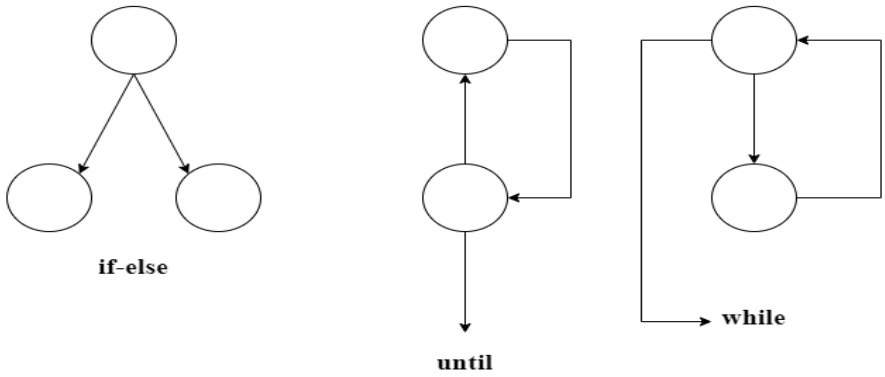


Рисунок 1.3 — Потокові графи операторів в програмуванні

ж) циклометрична складність. Це міра логічної складності програмного забезпечення, яка використовується для визначення кількості незалежних шляхів. Тобто для графа G, його циклометрична складність V(G). Розрахунок V(G) відбувається 3 способами:

1) V(G) = P + 1, де P — кількість предикатних вузлів у потоковому графі;
2) V(G) = E – N + 2, де E — кількість ребер, а N — загальна кількість вузлів;
3) V(G) = кількість неперетинних областей на графіку.

Наприклад, розрахуємо V(G), для графа G, що зображено на рисунку 1.4.

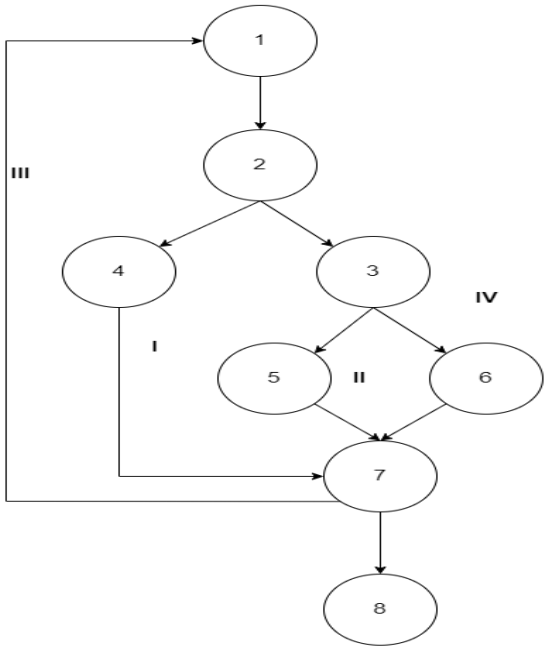


Рисунок 1.4 — Прикладний граф G

Використовуючи будь-який вище описаний спосіб розрахунку V(G), отримуємо, що V(G) = 4. Тобто кількість незалежних шляхів — 4. Приклад розрахування:

1) перший шлях: 1 – 2 – 4 – 7 – 8;
2) другий шлях: 1 – 2 - 3 - 5 - 7 – 8;
3) третій шлях: 1 – 2 – 3 – 6 – 7 – 8;
4) четвертий шлях: 1 – 2 – 4 – 7 – 1 – 2 – 4 – 7 – 8.

а) циклічне тестування. Циклічне тестування дуже важливе, тому що цикли широко використовуються й лежать в основі багатьох алгоритмів. Помилки часто виникають на початку та в кінці циклів. Цикли розділяють на 3 види:

1) прості цикли. Для простих циклів розміром n тестові випадки розроблені таким чином:
   - повністю пропустить цикл;
   - тільки один прохід через петлю;
   - 2 пропуски;
   - m проходить, де m < n;
   - n-1 і n+1 проходів.
2) вкладені цикли. Для вкладених циклів всі цикли мають мінімальну кількість, починаючи з внутрішнього циклу. Простий циклічний тест виконується у самому внутрішньому циклі й працює зовні, доки не будуть перевірені всі цикли;
3) пов'язані цикли. Це незалежні цикли один за одним. До кожного використовується простий повторний тест. Якщо вони є незалежними, розглядаємо їх як вкладені.

Переваги тестування білої скриньки:

– тестування білої скриньки дуже ретельне, оскільки перевіряється весь код та структури;

– це оптимізує помилки видалення коду та допомагає видалити надлишкові рядки коду;

– легко автоматизувати.

Недоліки тестування білої скриньки:

– головний недолік у тому, що такі тестування дуже дорогі;

– редизайн коду та переписування коду вимагають перезапису тестових випадків;

– тестувальники вимагають глибоких знань коду та мов програмування, на відміну від тестування методом чорної скриньки;

– існуючий код тестується, тому не можу виявити відсутню функціональність.

1.2.5 Тестування сірої скриньки

Це метод тестування програмного продукту або програми, що використовує часткове знання внутрішньої структури програми. Метою тестування сірої скриньки є пошук та виявлення дефектів, пов'язаних з неправильною структурою коду або неправильним використанням програми [10].

Цей процес зазвичай визначає контекстуальні помилки, пов'язані з вебзастосунком. Тож розширює область тестування, зосереджуючись на всіх рівнях системи. Тестування сірої скриньки поєднує тестування білої та чорної скриньок, це надає можливість частково розуміти внутрішню структуру коду, оскільки біла скринька розуміє внутрішню структуру коду, а чорна скринька — ні.

При розробці програмного забезпечення, тестування сірої скриньки дає можливість протестувати обидві сторони програми, рівень подання та частини коду. Це переважно корисно під час інтеграції та тестування на проникнення.

Приклад тестування сірої скриньки: при тестуванні вебсайту на наявність посилань, непрацюючих посилань тощо, коли тестувальники стикаються з проблемами з цими посиланнями, вони можуть швидко змінити HTML-код і провести тестування в режимі реального часу.

Тестування сірої скриньки проводиться з наступних причин:

– поєднує в собі переваги тестування чорної білої скриньки;

– зменшує накладні витрати на тривалий процес тестування функціональних та нефункціональних типів;

– надає розробникам багато часу для виправлення помилок;

– тестування виконується з погляду користувача, а не з погляду розробника.

1.3 Автоматизоване тестування

Автоматизоване тестування — це техніка тестування ПЗ, яка виконується за допомогою спеціальних інструментальних засобів для перевірки працездатності функціонала проєкту [11]. Даний спосіб містить в собі написання тестових сценаріїв, що дозволяє відтворювати їх безліч разів. Крім того, процес виконання автоматизованого тесту має можливість аналізувати прогнозовані та фактичні результати виконання тесту. Для IT-компаній це найефективніший метод тестування продукту, оскільки IT-ринок стрімко росте з кожним днем, то часу на ручне тестування може не бути. Проте слід поділяти, що потрібно автоматизувати, а чого не варто. Виділимо тестові випадки, які необхідно автоматизувати:

– ключові моменти для бізнесу;

– багаторазове використання функціонала проєкту;

– довготривалі щодо виконання тестові сценарії.

Вони є важливими для проєкту, оскільки їх автоматизування економить чималі ресурси проєкту. Проте слід пам’ятати, що не всі тестові випадки потрібно автоматизовувати, серед них:

– тестовий сценарій, який виконується одноразово;

– функціонал, до якого критерії та вимоги постійно змінюються.

Процес створення автоматизованих тестів є важливим, адже від правильного аналізу та підготовки сценаріїв залежить ефективність виконання таких тестів. Тож процес створення автоматизованих тестів виділяє три основні етапи:

а) планування, перед виконанням тестових сценаріїв слід підготувати середовище до стану «нового» для отримання актуальних даних під час тестування. Крім того, слід завчасно підготувати вхідні дані. Дотримання цього етапу забезпечує якість виконаних тестів;

б) виконання. Маючи підготовлене середовище, необхідно згідно з вимогами створити тестові сценарії та керувати їхнім процесом таким, як: запустити, редагувати, за необхідністю видалити;

в) звітування, результат виконання тестового сценарію повинен звітуватися наступними форматами на вибір: відеозапис, знімки екрана, log файлу.

Для автоматизації вище описаних тестових випадків є достатньо типів автоматизованого тестування, які допомагають вирішувати поставлене питання, серед них:

– модульний тест;

– димовий тест;

– інтеграційний тест;

– регресійний тест;

– API тест;

– функціональний тест;

– тест чорної скриньки;

– тест графічного інтерфейсу користувача;

Серед описаних типів автоматизованого тестування, для IT-компаній, котрі розробляють вебзастосунки цікавить тестування графічного інтерфейсу користувача. Цей тип можна вважати останнім серед тестування при завершенні розробки програмного забезпечення. Оскільки при його виконанні перевіряється чи відповідає наявний проєкт [12]:

– відповідності розробленого функціонала до інтерфейсу;

– коректність роботи розробленого інтерфейсу;

– перевірка всього розробленого вебзастосунку.

Тож використання даного типу автоматизованого тестування є актуальним для виконання даної роботи. Підсумовуючи, наведемо переваги автоматизованого тестування:

– автоматизація дозволяє охоплювати більше тестових випадків;

– швидке виконання тесту;

– виконання тесту з різними вхідними даними;

– менше використання ресурсів проєкту порівняно з ручним тестуванням.

1.4 Огляд існуючих рішень автоматизованого тестування графічного інтерфейсу користувача

Розглянувши існуючі рішення для реалізації програмного забезпечення автоматизованого тестування графічного інтерфейсу користувача, виділяють чотири підходи, котрі виділяються найефективнішими:

– інструментальні засоби запису та відтворення;

– створення тестових сценаріїв;

– тестування на основі таблиць;

– тестування на основі ключових слів.

Критерієм вибору правильного підходу залежить від технічної частини ПЗ. Тож спираються на наступні ключові моменти:

– легкість в інтеграції;
– низький поріг входження;
– швидкість виконання;
– масштабованість;
– підтримка.

1.4.1 Інструментальні засоби запису та відтворення

Це техніка, яка використовує спеціальні інструментальні засоби для автоматизованого тестування графічного інтерфейсу користувача без створення програмним кодом тестових сценаріїв. Мета цього способу тестування — записати дії тестувальника, які виконуються вручну. Тобто, коли тестувальник виконує певні дії над вебзастосунком вручну, інструмент фіксує ці дії та автоматично перетворює їх в сценарій. Наприклад, тестувальник вводить вхідні дані в поле в той час інструментальний засіб фіксує:

а) яке поле було залучено;
б) які дані були введені.

З цих двох дій інструментальний засіб формує остаточний тестовий сценарій.

У роботі [13] автори Li, Kanglin і Mengqi Wu дослідили ефективність використання такого способу тестування для власного готового вебзастосунку. Перш за все вони визначили, що для того, щоб використовувати інструмент запису та відтворення не потрібно знати мов програмування. Тож для користування таким методом тестування достатньо вміти користуватися загальними функціями комп’ютера, а самим інструментальним засобом користуватися доволі легко. Для малих стартап-проєктів — це найлегший перехід від ручного до автоматизованого тестування, оскільки не потрібно

наймати тестувальника через те, що записувати та відтворювати тестові сценарії зможе звичайний розробник або менеджер проєкту.

Проте, у роботі [14] автори Oliver Stadie і Peter M. Kruse використовували інструментальний засіб запису паралельно з розробкою вебзастосунку. Тестування було досить не ефективним способом, оскільки займало стільки ж часу, як і при ручному тестуванні. Це відбувалося через те, що застосунок, який розроблявся постійно змінювався, а отже потрібно було перезаписувати тестові сценарії.

Описаний спосіб тестування є легким у використанні для будь-якого проєкту, але не для всіх цей спосіб буде ефективним. Для готових рішень, які не будуть піддаватися суттєвим змінам у функціоналі, спосіб тестування запису та відтворення є доцільним, оскільки паралельно з записом тестового сценарію відбувається й ручне тестування, а процес запису є легким та не потребує додаткових знань у сфері тестування. Проте для проєктів, які знаходяться на стадії розробки, такий спосіб тестування не є доречним, оскільки при внесенні змін в програмне забезпечення необхідно всі тестові сценарії перезаписувати, що рівняється до ручного тестування.

1.4.2 Створення тестових сценаріїв

Процес тестування використовуючи даний метод починається після створення програмною мовою тест сценаріїв. Мета даного методу полягає описати програмним кодом тест сценарій, який потенційно може відтворити кінцевий користувач користуючись графічним інтерфейсом ПЗ. У роботах [15, 16] автори Mark Grechanik, Qing Xie та Chen Fu дослідили основи та ефективність даного методу тестування під час розробки власного вебзастосунку. Вони досягли наступних показників:

– правильний аналіз потенційних сценаріїв застосунку, гарантує, що необхідність внесення змін в тестовий сценарій мінімальна, крім випадків, якщо буде змінений чи доданий функціонал;

– для великого за обсягом проєкту, широкомасштабна перевірка ключового функціонала — гарантована. Оскільки за обсягом створений тест сценарій може бути будь-якого розміру, що дозволяє описувати чималу кількість функцій.

Саме ці показники показують реальну готовність розробленого проєкту, оскільки під час тестування видно весь масштаб виконаної роботи й правильність логічної побудови сценаріїв для користування.

В статті [17] автори Emil Borjesson та Robert Feldt висвітлили порівняння методів автоматизації між записом та відтворення й створенням тестових сценаріїв мовою програмування для вебзастосунку. Створювати тести першим методом, набагато простіше, оскільки достатньо натиснути кнопку запису та відтворити ручним способом тестовий сценарій. Інструментальний засіб виконає роботу за тестувальника та сформує всі виконанні кроки в програмний код. Якщо використовувати другий метод, час на розробку тестових сценаріїв займе більше ніж при тестуванні записом та відтворення, проте при внесенні змін в тестовий сценарій даним методом достатньо змінити програмний код в певному місці в порівнянні з методом записом та відтворення, де сценарій необхідно перезаписувати заново. Підсумовуючи, кожний метод тестування призначений для певного життєвого циклу ПЗ. Метод запису та відтворення підходить найефективніше використовується під час готового ПЗ. Для ПЗ, котре розробляється, доцільніше використовувати метод створення тест сценаріїв програмною мовою через можливість внесення змін без переписування всього сценарію.

1.4.3 Тестування на основі таблиць

Метод тестування на основі таблиць було описано в збірнику [18]. Автор Carl Nagle запропонував тестувати програмне забезпечення так, щоб дані виконання тесту зберігалися в форматі таблиці. Мета такого підходу полягає в розробці єдиного тестового сценарію, котрий буде відтворювати тести для всіх

тестових даних у таблиці та зберігати результати. Оскільки тестові дані зберігаються окремо від тестових сценаріїв, це дозволяє запускати тестовий сценарій з різною комбінацією вхідних даних.

Для отримання ефективного процесу тестування наведено наступні методи, котрих необхідно дотримуватися:

– використовувати реальні дані;
– позначати статус тесту позитивний або негативний.

У роботі [19] автори реалізували дану методику для вебзастосунку. Оскільки на час впровадження в застосунку були наявні регресійні тести, це не вплинуло на реалізацію тестування на основі таблиць. Тому був проведений дослід, чи зможуть регресійні тести використовувати вхідні дані з методу тестування на основі таблиць. В результаті, розробники досягли позитивних результатів досліду, під час регресійного тестування використовувалися різні набори вхідних даних, які були представлені у таблиці. Проте використання описаного підходу на реальних проєктах є затратним по ресурсах, оскільки таке тестування займає багато часу.

Описаний метод тестування підходить для перевірки тестового сценарію з різними вхідними даними, крім того, є можливість поєднання з регресійними тестами в проєкті. Проте розробляти такий метод доволі складно, оскільки необхідно знати, як розробляти мову сценаріїв та документацію методики, також складно підтримувати створені тести з розростанням проєкту.

1.4.4 Тестування на основі ключових слів

Метод тестування на основі ключових слів був розроблений інженером програмного забезпечення Danny R. Faught та представлена у статті «Keyword-Driven Testing» [20]. В даній статті було запропоновано створення сценаріїв, які будуть використовувати файли даних, що містять ключові слова для програми, яка тестується. Ці ключові слова описують послідовність набору дій, які необхідні для виконання певного кроку тестового сценарію. В кінцевому

результаті такий сценарій буде складатися з ключових слів високого та низького рівня, включаючи ключові аргументи, які створені для опису дій кроку.

Переваги використання даного методу у розробці програмного забезпечення полягають в:

– написання тестів виконується абстрактно, що дозволяє перевикористовувати створений тест;

– деталі тестового сценарію не доступні для звичайного користувача;

– легкий в розробці та розумінні, оскільки не потрібно знати додаткових інструментальних засобів.

У роботах [21, 22] автори реалізували даний підхід для тестування власного комп'ютерного додатку під час його розроблення. В результаті вони досягли наступних показників в тестуванні:

– уникнення дублікації коду;

– повторне використання тесту;

– зменшення ресурсів тестування порівняно з ручним тестуванням;

– легко розробляти.

Оскільки тестування на основі ключів впроваджувалось в застосунок під час його початку створення, то вже під завершення його розробки автори зіштовхнулися з проблемою підтримки даного методу тестування через розростання розробленого застосунку. Також слід зазначити, що чим більше тестів створювалися, тим важче ними було керувати чи прослідкувати їх результат.

В підсумку дана методика тестування підходить лише для маленьких застосунків, котрі розробляються малочисельною командою розробників. Оскільки впровадження тестування на основі ключів в масштабний проєкт викличе суттєві проблеми для його підтримки, незважаючи на легкість розробки таких тестів.

1.5 Постановка задачі

Метою магістерської дисертації є проєктування та розроблення програмного забезпечення для автоматизованого тестування графічного інтерфейсу користувача за допомогою використання спеціальних інструментальних засобів.

Методика тестування — автоматизоване тестування.

Тип тестування — тестування графічного інтерфейсу користувача.

Метод тестування — створення тест сценаріїв за допомогою мов програмування.

Для виконання задачі, були поставленні наступні завдання:

- аналіз предметної області;
- огляд існуючих методів та рішень;
- розробити архітектурне рішення, за допомогою якого буде легко впроваджувати дії інструментального засобу автоматизованого тестування без необхідності змін в інших рівнях архітектури;
- забезпечити різновид типів функцій подій;
- розробити вебзастосунок в якому можна керувати тестуванням. А саме: створення тест сценаріїв, створення вхідних даних, створення інформації про проєкт, котрий буде тестуватися;
- вебзастосунок повинен бути зручним для користування;
- оптимізувати серверну частину та базу даних, для швидкодії застосунку.

Висновки до розділу

Тестування в програмному забезпеченні відіграє головну роль на кожному етапі розробки та після виконання, для підтримки та перевірки працездатності

застосунку. Саме цей процес допомагає виявити дефекти, а в деяких випадках дослідити причину критичних помилок.

Автоматизоване тестування є найефективнішим способом для перевірки працездатності всього функціоналу проєкту. Адже це суттєва мінімізація використання ресурсів проєкту порівняно з ручним тестуванням. Розглянувши типи автоматизованого тестування, для розробки було обрано тестування графічного інтерфейсу користувача. Оскільки даний тип є завершальним в стадії тестування розробленого ПЗ та найважливішим для ІТ-компанії, бо виявлення помилок перед випуском проєкту на ринок може вплинути на його успіх.

Розглянуті методи реалізації тестування графічного інтерфейсу застосунку мають свої переваги та недоліки. Метод запису та відтворення легкий в користування, а додаткових знань в користуванні не потребує, проте якщо в застосунку будуть регулярно вноситися зміни в функціонал, то даний метод буде неефективним через необхідність перезаписування тест сценаріїв. Метод створення тестових сценаріїв мовою програмування, для його реалізації необхідно знати інструментальний засіб, який надає функціонал для тестування. Дану методику можна використовувати під час розробки ПЗ, оскільки внесені зміни в основний застосунок потребують малих змін в створений тест сценарій. Метод тестування на основі таблиць є ефективним для малого за функціоналом проєктів, оскільки для великого проєкту такий метод важко підтримувати, не дивлячись на його суттєву перевагу — виконання тест сценарію з усіма заданими вхідними даними. Метод тестування на основі ключових слів має ті ж проблеми, що й попередник, його найефективніше застосування, це проєкт, котрий малий за функціоналом та потребує лише пару тест сценаріїв.

В підсумку було поставлено задачу для успішного виконання даної роботи, визначено мету, методику, тип та метод тестування, котрий буде розроблений в програмному рішені.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Проєктування широко функціонального програмного забезпечення вимагає його розбиття на модулі. Це необхідно для спрощення проєктування застосунку та його розробки. Модуль повинен виконувати лише свій функціонал згідно з призначенням. Правильне проєктування програмного забезпечення дозволить:

– розгортати застосунок в будь-якому хмарному середовищі;

– комунікація між модулями легко налаштовується;

– незалежність модулів, дозволяє описувати логіку модуля згідно з його сутністю;

– масштабування, дозволяє добавляти нові модулі без суттєвих затрат ресурсів.

Оскільки для вирішення поставлених задач необхідно зберігати дані, то виникає потреба в модулі, який буде виконувати цю роль. Називають такий модуль, базою даних. Даний модуль буде відповідати за наступні задачі:

– зберігання даних;

– керування даними: добавити, оновити, видалити;

– розробка скриптів для заповнення даних;

– розробка скриптів для керування даними.

Для керування даними необхідний інший незалежний модуль. Назвемо його серверна частина програмного забезпечення. Цей модуль буде виконувати наступні задачі:

– опис логіки згідно з поставленими вимогами;

– розробка алгоритмів;

– отримання та передача даних.

Для відображення даних користувачу є потреба ще в одному модулі. Назвемо його клієнтська частина. Цей модуль буде відповідати за наступні задачі:

– відображення даних на графічному інтерфейсі користувача;

– оброблення вхідних даних від користувача.

Підсумовуючи, для проєктування програмного забезпечення необхідно спроєктувати три послідовних модулі:

а) модуль бази даних;

б) модуль серверної частини;

в) модуль клієнтської частини.

2.1 Визначення сутностей

Згідно з оглядом предметної області було визначено, що сфера тестування є достатньо об’ємною, оскільки поділяють багато типів та підходів для їх реалізації. Тож визначимо основні сутності, які необхідні для проєктування та розробки програмного забезпечення:

а) функція подій. Сутність для опису, яку дію необхідно відтворити. Наприклад: відправити запит чи вибрати поле;

б) компаньйон. Сутність, яка є групою певної кількості функцій подій. Наприклад: натиснути на елемент, заповнити поле, зробити знімок. Ці функції подій можна згрупувати;

в) тестовий сценарій. Сутність, яка призначення для опису черги функцій подій під час виконання тесту;

г) проєкт. Сутність, яка призначена для заповнення інформації про вебзастосунок та його розроблені тестові сценарії.

Оскільки функції подій можуть виконувати різні дії над вебзастосунком, слід розподілити їх по типах, для спрощеного проєктування та розробки програмного забезпечення:

– технічний тип. Це дія, яка відтворюється користувачем технічно. Наприклад: виділити фрагмент елементу, скопіювати та вставити в поле;

– графічного інтерфейсу тип. Це дія, яка виконує графічні операції. Наприклад: знімок екрана;

– REST тип. Це дія, яка виконує запити на сервер вебзастосунку;

– SQL тип. Це дія, яка виконує операції з базою даних.

2.2 Проєктування бази даних

Для проєктування бази даних згідно з вимогами поставлених задач слід дотримуватися наступних вимог для того, щоб після виконання проєктування не повертатися та суттєво змінювати структуру бази даних:

– масштабованість. Розроблена структура не повинна тісно пов’язана зі сутностями для уникнення складностей під час добавлення нових таблиць та зв’язків;

– зрозуміла. Нагромадження таблиць атрибутами викликають суттєві проблеми під час підтримки бази даних. Для уникнення нагромадження слід розбивати великі таблиці на під таблиці добавляючи зв’язки між ними;

– безпека. Вносити зміни в базу даних повинен тільки адміністратор. Користувач має можливість тільки отримати дані;

– не надлишковість. Для уникнення випадків, коли видаляється елемент даних з однієї таблиці, він зникає в кожній;

– несуперечливість. Тип даних має відповідати їх реальному значенню.

Мета проєктування бази даних — це перетворення описаної предметної області в середовище бази даних. Цей процес відбувається поступово, спочатку описується інфологічна модель бази даних. Мета інфологічного моделювання описати на діаграмі сутності та зв’язки між ними. На рисунку 2.1 зображена інфологічна модель для сутностей, які пов’язані з функцією подій. Це сутності: типів функцій подій, результати їх виконання. Зв’язки між ними: один до одного, один до багатьох.

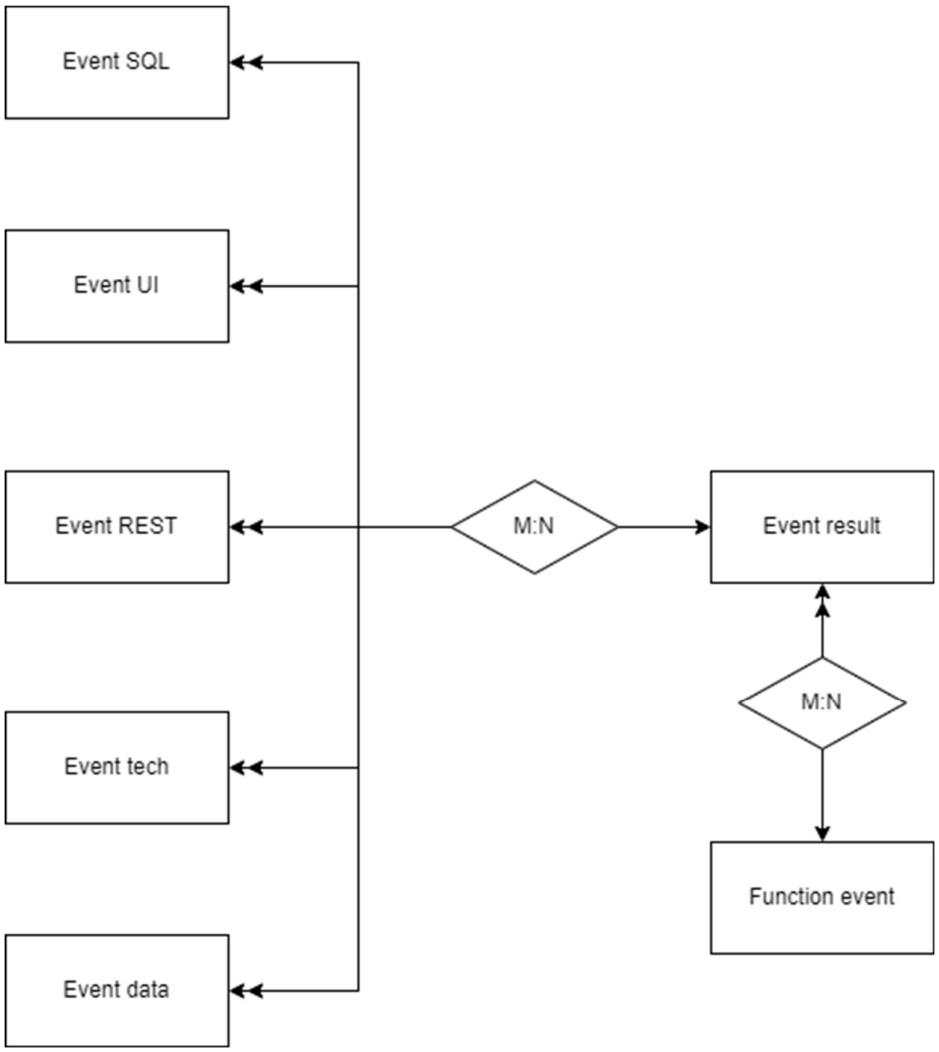


Рисунок 2.1 — Інфологічна модель функцій подій

2.3 Проєктування серверної частини

Спроєктувавши базу даних та визначивши сутності та їх особливості, наступним кроком є проєктування серверної частини. Для проєктування сервера програмного забезпечення використовують архітектурні підходи, такі як:

- мікросервісна архітектура;
- клієнт-сервер архітектура;
- модель відображення архітектура;
- багаторівнева архітектура.

Важливим критерієм вибору архітектури для даного програмного забезпечення є розробка кожного рівню чи модулю незалежно від інших. Тож вибір припав на багаторівневу архітектуру, котра дозволяє впроваджувати рівні незалежно, що відповідає вимогам масштабованості. За основну структуру багаторівневої архітектури було обрано трьох рівневу архітектуру, яка розподіляє кожний рівень по задачах, розглянемо їх:

– рівень доступу до даних, керує даними, які надає база даних;

– рівень додатків, обробляє дані згідно з бізнес вимог;

– рівень презентації, надає оброблені дані клієнтській частині або приймає від неї та перетворює у потрібний формат.

Тож кожна сутність програмного забезпечення повинна бути описана на кожному рівні трьох рівневої архітектури.

Проте виникає проблема з масштабованістю, оскільки при добавлені нових функцій подій, їх необхідно впроваджувати на кожному рівні архітектури. Проблема такого підходу полягає в не гнучкості розробки та затратності по ресурсах. Оскільки при добавленні нових двадцять функцій подій, необхідно їх всі описувати на кожному рівні та можливо рефакторити інші методи чи класи програмного забезпечення, що суттєво впливає на час розробки.

Для вирішення цієї проблеми, пропонується винести функції подій, як окремий рівень архітектури програмного забезпечення. Це дозволить впроваджувати нові функції подій без внесення змін на інших рівнях. Для того, щоб програмне забезпечення розуміло, які існують функції подій, необхідно на рівнях доступу до даних, додатків, презентації реалізувати абстрактний клас функції подій. Рівень доступу до даних буде взаємодіяти з рівнем функцій подій, зчитуючі, які існують функції, тим самим виконуючи свою поставлену задачу по керуванню даними. На рисунку 2.2 зображено чотирьох рівневу архітектуру програмного забезпечення.

Рисунок 2.2 — Чотирьох рівнева архітектура програмного забезпечення

Висновки до розділу

У даному розділі програмне забезпечення було розбите по модулях: база даних, серверна частина, клієнтська частина. Кожний модуль працює незалежно та виконує задачі згідно з вимогами.

Було описано важливі сутності предметної області та визначені їх особливості. Для сутності функцій подій є потреба розподілу їх по типах, для спрощеного розуміння та розробки.

Під час проєктування бази даних, було оглянуто основні вимоги для проєктування. Також було розроблено діаграму інфологічної моделі для функцій подій.

Проєктування серверної частини потребувало в новому рішенні архітектурного підходу для автоматизованого тестування. Оскільки розроблення на основі трьох рівневої архітектури призвело б до розгалуження всієї системи, що значно вплинуло б на процес розробки та ефективність. Було розроблено новий архітектурний рівень, котрий немає цих недоліків.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вибір технологій

Правильний вибір технологій для розробки програмного забезпечення гарантує: ефективність, стабільність, швидкодію розробленого рішення та легкість використання технологій під час розробки. Тож для вибору технології слід дотримуватися наступних критеріїв:

– довготривале рішення, гарантує працездатне рішення перевірене роками;

– підтримка різних версій технології;

– ефективність використання підтверджується іншими розробниками.

3.1.1 Вибір технологій для бази даних

Для реалізації бази даних було обрано реляційну систему керування даними, котра програмується мовою SQL. Особливістю системи є організація даних у вигляді таблиць та зв’язків між ними. Програмним забезпеченням для керування базою даних обрано PostgreSQL [23]. Особливістю обраного рішення є відмовостійкість до збоїв, тим самим забезпечує цілісність даних. Розробка в PostgreSQL є зручною, оскільки керування базою даних відбувається в веббраузері.

3.1.2 Вибір технологій для серверної частини

Серверна частина – це основна логіка обробки функцій, даних програмного забезпечення. Ефективність виконання котрих залежить від вибору технологій.

Для розробки клієнтської частини було обрано наступні технології:

– Java, мова програмування для розроблення серверної частини програмного забезпечення [24];

– Spring Framework, широко функціональний фреймворк, котрий надає основні послуги для розробки програмного забезпечення згідно об’єктно-орієнтованому програмуванню [25];

– Maven, інструментальний засіб для керування залежностями та збірки проєкту [26];

– Selenium, фреймворк, котрий надає використання функцій автоматизованого тестування [27];

– ChromeDriver, інструментальний засіб для відтворення запрограмованих тест сценаріїв у веббраузері [28].

3.1.3 Вибір технологій для клієнтської частини

Графічний інтерфейс користувача забезпечує можливістю користування розроблених функцій серверної частини. Розробка привабливого та зрозумілого інтерфейсу надає користувачу позитивні враження, що є важливим, як для клієнта, так й для IT-компанії. Тож вибір технологій для реалізації графічного інтерфейсу мають відповідати сучасним дизайнам та можливістю для їх реалізації.

Для розробки клієнтської частини було обрано наступні технології:

– HTML, мова гіпертекстової розмітки, яка використовується для опису DOM-елементів сторінки [29];

– LESS, мова стилів для оформлення HTML-сторінок [30];

– Typescript, мова програмування за допомогою якої описується функціональність графічного інтерфейсу [31];

– Angular, фреймворк для розробки вебзастосунків, котрий надає можливість розробляти кросплатформенні рішення [32];

– Angular Material, бібліотека шаблонів компонентів графічного інтерфейсу [33];

– SockJS, бібліотека для реалізації зв’язку між клієнтською та серверною частиною за допомогою вебсокетів [34].

3.2 Розробка бази даних

Виконавши інфологічне проєктування бази даних, наступним кроком є фізичне проєктування. Мета фізичного проєктування є розробка скриптів в СУБД згідно сформованій моделі та зв’язків між ними. Дані скрипти програмно описують структуру бази даних за допомогою мови SQL.

В таблицях 3.1-3.7 наведено структури таблиць, які необхідні для функцій подій. ER-діаграму схеми бази даних наведено у додатку А.

Таблиця 3.1 — Опис таблиці function_event

Назва стовпця	Опис	Тип даних	Довжина	Первинний ключ	Зовнішній ключ
event_id	Ідентифікатор	INT	MAX	X	
class	Функціональний опис	VARCHAR	256		
description	Загальний опис події	TEXT	MAX		

Таблиця 3.2 — Опис таблиці event_result

Назва стовпця	Опис	Тип даних	Довжина	Первинний ключ	Зовнішній ключ
result_id	Ідентифікатор	INT	MAX	X	
event_id	Зовнішній ключ до події	INT	MAX		X
test_case_result_id	Зовнішній ключ до тесту виконання	INT	MAX		X
status_id	Зовнішній ключ до статусу	INT	MAX		X
start_date	Початок виконання	DATE	MAX		
end_date	Кінець виконання	DATE	MAX		
report	Звіт виконання	TEXT	MAX		

Таблиця 3.3 — Опис таблиці event _data

Назва стовпця	Опис	Тип даних	Довжина	Первинний ключ	Зовнішній ключ
data_id	Ідентифікатор	INT	MAX	X	
event_result_id	Зовнішній ключ до результату функції подій	INT	MAX		X

Продовження таблиці 3.3

key	Ключ	VARCHAR	128		
value	Опис значення	TEXT	MAX		

Таблиця 3.4 — Опис таблиці event_tech

Назва стовпця	Опис	Тип даних	Довжина	Первинний ключ	Зовнішній ключ
tech_id	Ідентифікатор	INT	MAX	X	
event_result_id	Зовнішній ключ до результату функції подій	INT	MAX		X
key	Ключ	VARCHAR	128		
value	Опис значення	TEXT	MAX		

Таблиця 3.5 — Опис таблиці event_rest

Назва стовпця	Опис	Тип даних	Довжина	Первинний ключ	Зовнішній ключ
rest_id	Ідентифікатор	INT	MAX	X	
event_result_id	Зовнішній ключ до результату функції подій	INT	MAX		X
request	Тіло REST запиту	TEXT	MAX		

Продовження таблиці 3.5

response	Тіло REST відповіді	TEXT	MAX		
code	Статус код REST	INT	MAX		

Таблиця 3.6 — Опис таблиці event_ui

Назва стовпця	Опис	Тип даних	Довжина	Первинний ключ	Зовнішній ключ
ui_id	Ідентифікатор	INT	MAX	X	
event_result_id	Зовнішній ключ до результату функції подій	INT	MAX		X
path	DOM шлях до об’єкту	VARCHAR	512		

Таблиця 3.7 — Опис таблиці event_sql

Назва стовпця	Опис	Тип даних	Довжина	Первинний ключ	Зовнішній ключ
sql_id	Ідентифікатор	INT	MAX	X	
event_result_id	Зовнішній ключ до результату функції подій	INT	MAX		X

Продовження таблиці 3.7

environmet	Посилання на розташування БД	VARCHAR	512		
user	Логін користувача	VARCHAR	128		
query	SQL запит	VARCHAR	512		

Наступним кроком є створення тестових даних, котрі необхідні для тестування програмного забезпечення під час розробки. Слід зазначити, що тестові дані повинні бути реалістичними для коректної роботи застосунку.

Реалізувавши програмно структуру бази даних в СУБД та заповнивши її тестовими даними, наступним кроком є розробка запитів до таблиць. Запити необхідні для отримання даних, котрі зберігаються в базі даних та можуть бути представлені у вільному вигляді таблиці.

Приклад запитів, які були розроблені:

– findFunctionEventsByClass, запит для отримання функцій подій згідно заданого типу;

– saveFunctionEventRest, запит для зберігання результату виконання функцій подій типу REST;

– updateDataParameters, запит для оновлення вхідних даних.

Розробивши запити, їх слід протестувати на тестових даних для того, щоб перевірити працездатність та час виконання. Оскільки структура бази даних проєктувалася згідно трьом формам нормалізації, час виконання об’ємних запитів може бути тривалим. Під час тестування, було виявлено, що розроблений запит формування черги функцій подій виконувався 4,6 секунди. Отриманий час не є критичним для застосунку, проте при більшій кількості даних, цей час виконання буде зростати. Тому для підтримання ефективності застосунку було

пожертвувано нормалізацією до другої форми. В підсумку час виконання запиту становить 3,2 секунди, що на 1,4 секунди менше ніж при третій формі.

3.3 Розробка серверної частини

Розробивши структуру бази даних та заповнивши її тестовими даними, наступним кроком для розробки програмного забезпечення є розробка серверної частини.

Серверна частина відповідає за весь функціонал застосунку, обробляючи дані в базі даних та передачі їх на клієнтську частину. Тож для безперебійної роботи серверної частини слід дотримуватися наступних підходів під час розробки програмного забезпечення:

а) об’єктно-орієнтоване програмування, це парадигма програмування сутність якої полягає, що кожний клас чи об’єкт повинен відповідати згідно своєї сутності [35];

б) SOLID принципи — п’ять принципів об’єктно-орієнтованого програмування, котрі допомагають розробляти програмне забезпечення більш універсально та зручно [36];

в) DRY — підхід в програмуванні котрий називають «Не повторюй себе». Сутність підходу уникати повторного використання коду [37].

3.3.1 Перший рівень архітектури

В даному рівні розробляються функції подій використовуючи фреймворк Selenium, котрий надає розроблені інструментальні засоби для автоматизованого тестування. Було розроблено анотацію «@FunctionEvent», котра описує метадані функцій подій, наприклад: назва, параметри, тип. В таблиці 3.8 розглянуті приклади створених функцій подій.

Таблиця 3.8 — Створені функції подій

Назва	Призначення	Тип
Click on element by Id	Натиснути на елемент DOM-сторінки по ідентифікатору	Технічний
Input Data by Xpath	Заповнити дані елементу DOM-сторінки по Xpath	Технічний
Copy label by Id	Скопіювати тег елементу DOM-сторінки по ідентифікатору	Технічний
Send REST	Відправити REST запит	REST
Snapshot	Зробити знімок екрану	UI
Execute SQL request	Виконання SQL запиту	SQL

В результаті завершення розробки першого рівня архітектури було створено 18 функцій подій. Приклад програмної реалізації функцій подій різних типів описано у додатку Г.

3.3.2 Другий рівень архітектури

В даному рівні описується обробка отриманих даних з бази даних та відправлення відповіді на третій рівень. Крім того обробляються функції подій, котрі створені на першому рівні. Кожний клас відповідає за свою сутність. В таблиці 3.9 наведено опис основних класів другого рівня архітектури.

Таблиця 3.9 — Опис класів другого рівня архітектури

Клас	Призначення	Метод
FunctionEvent	Обробка даних функцій подій	Find() – пошук функції; Update() – оновлення функції; getType() – отримати тип фунції; init() – ініціалізувати функцію; connect() – під'єднати функцію до БД; getMeta() – отримати метадані.
Companion	Об'єднання функцій подій	Save() – зберегти об'єднання; FindById() – знайти об'єднання по ідентифікатору; FindAll() – знайти всі об'єднання; deleteById() – видалити об'єднання по ідентифікатору; updateById() – обновити об'єднання по ідентифікатору.
Step	Формування черги для функцій подій	Save() – записати функцію; Update() – оновити чергу; findById() – знайти чергу по ідентифікатору.
Scenario	Формування тестового сценарію	Create() – створити сценарій; Update() – оновити сценарій; Delete() – видалити сценарій; findById() – знайти сценарій по ідентифікатору; findProjects() – знайти проєкти, які використовують сценарій.

Всього було розроблено 17 класів для обробки даних кожної сутності проєкта.

3.3.3 Третій рівень архітектури

Отримані дані з другого рівня архітектури, котрі містять в собі загальні дані програмного забезпечення та функції подій, обробляються згідно з вимогами бізнес-логіки в третьому рівні.

Під час обробки, дані можуть мутуватися в інші за типом об’єкти або доповнюватися додатковими властивостями. Саме, тому бізнес-логіка описує вимоги, такі як, задання правил спілкування об’єктів між собою

Було розроблено всі необхідні класи сервіси, для описаних класів в другому рівні архітектури.

Оброблені об’єкти на третьому рівні передаються на четвертий рівень архітектури серверної частини.

3.3.4 Четвертий рівень архітектури

Даний рівень виступає комунікатором між клієнтською частиною та бізнес-логікою проєкту. Всі сутності програмного забезпечення мають власний комунікатор. Класи даного рівня були розроблені згідно RESTful підходу та позначені відповідною анотацією «@Controller». Ці класи слухають клієнтську частину, котра передає тіло запиту. Комунікатор формує тіло запиту згідно з об’єктами описаних сутностей проєкту, а далі передає об’єкти до бізнес-логіки застосунку. У відповідь до клієнтської частини комунікатор надає тіло відповіді або помилку.

Якщо тіло запиту передано не в правильному форматі, було створено допоміжний клас для обробки помилок та позначений відповідною анотацією «@ControllerAdvise». Функції даного класу успадковують класи-сутності обробки помилок, де обробляються помилки згідно з визначеними правил для кожної з сутностей.

3.4 Розробка клієнтської частини

Розробивши всі попередні модулі програмного забезпечення залишається розробка клієнтської частини, завдяки якій користувач зможе використовувати розроблений функціонал програмного забезпечення.

Першим кроком є налаштування клієнтської частини з серверною частиною через конфігураційний файл Maven. Це надає змогу комунікації між модулями програмного забезпечення.

Наступним кроком є опис сутностей, як об’єкт. Це необхідно для коректного керування типом даних. Маючі налаштований зв’язок з серверною частиною та розроблені моделі для вказування типів даних, що приходять. Потрібно реалізувати клас, котрий буде керувати запитами для отримання чи відправки. В Angular такий клас називають — сервіс. В даному класі описуються необхідні HTTP методи, такі як: POST, PUT, PATCH, DELETE, GET.

Маючи всі необхідні підготовлені засоби для отримання та обробки даних клієнтської частини, наступним кроком є розробка логічних компонентів. Логічний компонент — це клас, котрий описує логіку сутності застосунку. Даний клас використовує методи, що представленні у сервісному класі для отримання даних. Ці дані обробляються згідно з вимогами для візуалізації їх на екрані користувача. Кожний такий клас також містить HTML-шаблон, в котрому описується графічні елементи.

Оскільки кількість логічних класів може бути велика, у випадку даної роботи, це 17 компонент, доцільно розбити їх на модулі. В таблиці 3.10 опис модулів, їх призначення та компоненти, які зберігаються.

Таблиця 3.10 — Опис модулів клієнтської частини

Модуль	Призначення	Компоненти
Бібліотека	Зберігання компонентів, які виконують функції подій та їх вхідні дані	Function Event; Companion; Library.
Проєкт	Зберігання проєктів, тест сценаріїв, результатів	Project; Test scenario; Data parameters; Results.
Користувач	Зберігання загальної інформації користувача	User; Registration; Profile.

3.5 Експериментальне дослідження

Для визначення ефективності розробленого програмного забезпечення, будемо порівнювати процес добавлення функцій подій для трьох рівневої та чотирьох рівневої архітектури. Виконання порівняння буде відбуватися за наступними критеріями:

- витрачений час на добавлення нової функції події;
- масштабування проєкту;
- легкість виконання.

Також важливо враховувати фактори, які впливають на добавлення функцій подій. Якщо бажана функція подій є складна для добавлення, це може зайняти більше часу, оскільки для її розробки потрібно ознайомитися з додатковою інформацією, яку надає фреймворк у своїй документації.

Розглянемо добавлення функцій подій для обраних архітектур. Для кожної з архітектур сформуємо таблицю, в котрій будемо вказувати показники під час проведення досліду. Опишемо показники та спосіб вимірювання:

– час реалізації, показник котрий подає відомість скільки хвилин витратили для реалізації функції подій. Вимірюється у хвилинах;

– час адаптування інших функцій, показник відображає скільки часу необхідно для адаптування існуючих функцій програмного забезпечення після добавлення нової функції подій. Вимірюється у хвилинах;

– тестування, показник для визначення скільки часу необхідно для перевірки правильної реалізації;

– виконання, показник, який визначає складність реалізації функції подій та її адаптування. Вимірюється за наступними показниками: легко, виникли певні труднощі, важко.

У таблицях 3.11-3.12 описано результати дослідження для трьох та чотирьох рівневої архітектури.

Таблиця 3.11 — Результати дослідження трьох рівневої архітектури

Назва	Час реалізації	Час адаптування інших функцій	Тестування	Виконання
Натиснути двічі на елемент по ідентифікатору	3 хвилини	42 хвилини	5 хвилин	Легко
Натиснути на елемент та ввести дані в поле	12 хвилин	72 хвилини	5 хвилин	Виникли певні труднощі
Натиснути на елемент та зробити знімок екрану	10 хвилин	120 хвилин	5 хвилин	Важко

Таблиця 3.12 — Результати дослідження чотирьох рівневої архітектури

Назва	Час реалізації	Час адаптування інших функцій	Тестування	Виконання
Натиснути двічі на елемент по ідентифікатору	5 хвилини	0 хвилини	5 хвилин	Легко
Натиснути на елемент та ввести дані в поле	15 хвилин	0 хвилини	5 хвилин	Легко
Натиснути на елемент та зробити знімок екрану	15 хвилин	0 хвилин	5 хвилин	Легко

Згідно з отриманими результатами дослідження сформуємо графік, в якому буде представлено порівняння добавлення функцій подій для кожної з архітектур. Об’єднаємо показники з кожної функції для демонстрації загального витраченого часу. Для показника «виконання» вагу показників сформуємо наступним чином: вага легкості – 1, вага виникли певні труднощі – 2, вага важко – 3. Графік порівняння зображено на рисунку 3.1.

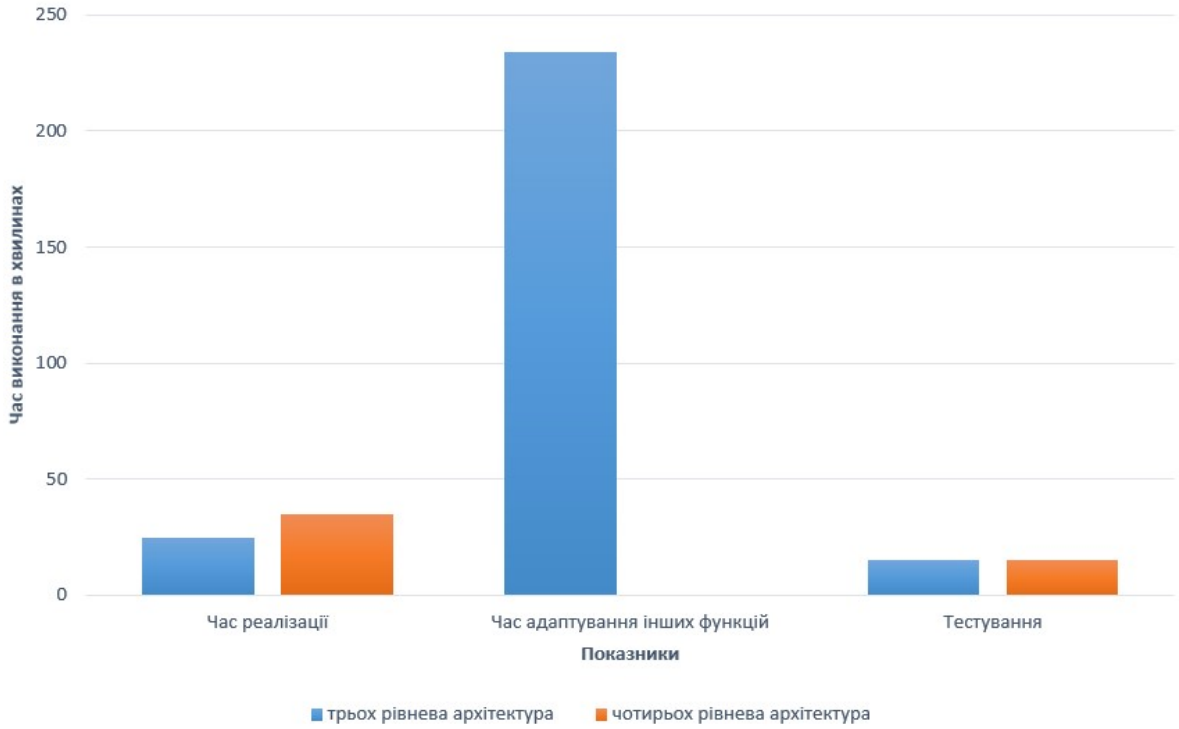


Рисунок 3.1 — Порівняння результатів добавлення функцій подій

Отриманий графік відображає наступні показники для реалізації трьох функцій подій.

Трирівнева архітектура:

– час реалізації займає 25 хвилин;

– час адаптування займає 234 хвилини;

– тестування займає 15 хвилин;

– вага складності виконання 6.

Чотирьох рівнева архітектура:

– час реалізації займає 35 хвилин;

– час адаптування займає 0 хвилини;

– тестування займає 15 хвилин;

– вага складності виконання 3.

Чотирьох рівнева архітектура займає більше часу для реалізації функцій подій порівняно з трьох рівневою. Проте час для адаптування інших функцій подій для трьох рівневої архітектури становить 234 хвилини, в той час для

чотирьох рівневої відсутня потреба для адаптації. Тестування в обох випадках займає однаковий час. В підсумку для повної реалізації функцій подій трьох рівневої архітектури необхідно 274 хвилини та вага складності реалізації 6. В той час, як для чотирьох рівневої показники для реалізації становлять 50 хвилин та вага складності 3.

Підсумовуючи згідно з отриманими результатами та аналізу, чотирьох рівнева архітектура найкраще підходить для реалізації. Загальний час виконання реалізації становить на 224 хвилини менше, в порівнянні з трьох рівневою архітектурою.

Висновки до розділу

У даному розділі було розглянуто: необхідні технології для розробки, розроблення кожної частини програмного забезпечення та проведення експериментального дослідження.

Серед обраних технологій: PostgreSQL, Spring, Java, Maven, Selenium. Angular, Typescript, HTML, LESS.

Під час розробки бази даних, було виявлено, що розроблена структура згідно нормалізації виконувала запит для отримання черги функцій подій — задовго, при тестових даних малого об’єму. Тож була проведена денормалізація, для пришвидшення виконання запиту.

Експериментальне дослідження виконувалось порівнюючі час для реалізації трьох функцій подій, трьох рівневої та чотирьох рівневої архітектури. Результати дослідження підтвердили ефективність розробленого програмного забезпечення, різниця між чотирьох рівневої та трьох рівневої архітектури становить 224 хвилини, а різниця ваги складності виконання — 3.

4 МАРКЕТИНГОВИЙ АНАЛІЗ СТАРТАП-ПРОЄКТУ

4.1 Опис ідеї проєкту

Автоматизація тестування графічного інтерфейсу користувача є ефективним способом для проведення тестування вебсайтів. В той час, як для проєкту — це є той спосіб тестування продукту, який економить час та ресурси.

Призначенням проєкту є розробка програмного забезпечення для автоматизації вебсайтів, що дозволяє створювати тестові сценарії та відтворювати їх автоматизовано.

Сутністю розробки проєкту є реалізація архітектурного підходу для розробки автоматизованого застосунку за допомогою інструментальних засобів тестування.

Цільовою аудиторією є розробники ПЗ та ІТ-бізнес. Для бізнесу даний проєкт, як ефективний інструмент, котрий виконує свою роботу та зекономлює ресурси виділені на проєкт. Для розробників, це не тільки застосунок в якому вони можуть створювати тестові сценарії, а мають можливість ознайомитися з запропонованою архітектурою та розробити власний застосунок для автоматизованого тестування згідно наявних потреб.

Основною перевагою проєкта є мінімізація використання ручного тестування графічного інтерфейсу користувача в проєктах, шляхом розробки програмного забезпечення автоматизованого тестування за допомогою спеціальних інструментальних засобів для цього.

Таблиця 4.1 деталізує зміст ідеї для стартап-проєкту ПЗ автоматизованого тестування, напрямки застосування цього проєкту та його переваги для потенційних користувачів.

Таблиця 4.1 — Опис ідеї стартап-проєкту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Розробка програмного забезпечення для автоматизованого тестування графічного інтерфейсу користувача за допомогою інструментальних засобів, котрі призначення для автоматизації	1. IT-компанії	Тестування проєкту буде відбуватися швидше та автоматизовано. При цьому ресурси проєкту займають лише мінімальних витрат. Для тестування ресурси необхідні тільки для створення тестових сценаріїв.
	2. IT-спільнота (розробники, тестувальники)	Розробник або тестувальник має можливість використання даного ПЗ для тестування власних вебзастосунків. Крім того, може ознайомитись з архітектурним підходом та розробити власне рішення, котре буде покривати необхідні поставлені вимоги

Продовження талиці 4.1

	3. Освітні заклади (для студентів та школяр, що вивчають програмування)	Студенти та школярі, що вивчають програмування та розробляють власні вебсайти, можуть тестувати свої рішення шляхом створення тестових сценаріїв та вказувати вхідні дані. Наприклад, це допоможе протестувати вебсайт на його коректну роботу з різними вхідними даними

Ідея стартап-проєкта полягає в розробці програмного забезпечення для автоматизованого тестування графічного інтерфейсу користувача. Мета ідеї надати користувачу можливість тестувати власний вебзастосунок використовуючи універсальні функції.

В таблиці 4.2 розглянуто порівняння стартап-проєкта з продукцією конкурентів. Порівняння здійснювалося згідно наступних критеріїв, котрі важливі для кінцевого користувача:

- структуризація по проєктам створених тестів;
- зберігання вхідних даних для тестів;
- одночасний запуск 2-ух і більше тестів;
- детальний графічний звіт виконання тесту;
- універсальність функцій подій;

– швидкість виконання тесту.

Проєкт, котрий відповідає всім вище описаним критеріям вважається найефективнішим. Рішення 1 має перевагу над моїм стартап-проєктом лише в одночасному запуску. Рішення 2 переваг немає, але має недоліки такі, як: структуризація по проєктам створених тестів, детальний графічний звіт виконання тесту, швидкість виконання тесту. Рішення 3 ідентичне в порівняння, проте воно пропонує метод тестування — запису та відтворення. Ефективність даного методу доступна лише на стадії завершення розробки ПЗ.

Таблиця 4.2 — Визначення сильних, слабких та нейтральних характеристик ідеї проєкту

№	Техніко-економічна характеристики ідеї	Продукція конкурентів				W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проєкт	Рішення 1	Рішення 2	Рішення 3			
1	Структуризація по проєктам створених тестів	+	-	-	+	+		
2	Зберігання вхідних даних для тестів	+	+	+	+		+	

Продовження таблиці 4.2

3	Одночасний запуск 2-ух і більше тестів	-	+	+	-			+
4	Детальний графічний звіт виконання тесту	+	-	-	+			+
5	Універсальність функцій подій	+	-	+	-			+
6	Швидкість виконання тесту	+	+	-	+			+

Таким чином, в результаті проведеного аналізу слабких, нейтральних та сильних сторін можна зробити висновок, що мій стартап-проєкт є цілком конкуретноспроможний, відповідаючи всім критеріям та маючи необхідний функціонал для проведення автоматизованого тестування вебзастосунків.

4.2 Технологічний аудит ідеї проєкту

В рамках технологічного аудиту ідеї проєкту необхідно виконати аналіз можливостей для розробки функціоналу та оцінити технології, що використовувались для його реалізації. У таблиці 4.3 представлено ідеї стартап-проєкту та технології реалізації цих ідей.

Таблиця 4.3 — Технологічна здійсненність ідеї проєкту

№	Ідея проєкту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Чотирьох-рівнева архітектура серверної частини ПЗ для додавання функцій подій	Фреймворк Spring	Наявна	Доступна
		База даних PostgreSQL	Наявна	Доступна
2	Поетапне проходження тестового сценарію	WebSocket	Наявна	Доступна
3	Графічний інтерфейс користувача	Фреймворк Angular	Наявна	Доступна

Згідно з таблиці, всі технології котрі потрібні для розробки ПЗ є в наявності та вільному доступі, тож наведемо обґрунтування їх використання:

– чотирьох-рівнева архітектура серверної частини ПЗ для додавання функцій подій — фреймворк Spring дозволяє гнучко побудувати архітектуру програмного забезпечення, в той час база даних PostgreSQL забезпечує надійне зберігання та надання даних;

– поетапне проходження тестового сценарію — технологія WebSocket дозволяє миттєво повідомляти користувачу, як тільки будуть отримані дані від серверної частини. Це дозволяє переглядати перші кроки виконаного тест сценарію, поки останні виконуються;

– графічний інтерфейс користувача — фреймворк Angular сучасний інструмент для розробки вебзастосунків, котрий надає можливості не тільки будувати сторінки, але й забезпечувати ефективність та швидкодію клієнтської частини програмного забезпечення.

4.3 Аналіз ринкових можливостей запуску стартап-проєкту

Альтернативні рішення тільки з’являються на ринку. Тож, динаміка ринку починає зростати, що сприяє на попит продукту. На даний момент для виходу на ринок відсутні вимоги, щодо сертифікації та стандартизації. В таблиці 4.4 розглянуто попередню характеристику для потенційного ринку стартап-проєкта.

Таблиця 4.4 — Попередня характеристика потенційного ринку

№	Показники стану ринку	Характеристика
1	Кількість головних гравців, од	5
2	Загальний обсяг продаж, грн./ум.од	Невідома
3	Динаміка ринку	Зростає
4	Наявність обмежень для входу	Відсутні
5	Специфічні вимоги до стандартизації та сертифікації	Відсутні
6	Середня норма рентабельності в галузі або по ринку, %	Невідомо

Розглянемо потенційних клієнтів стартап-проєкту, їх відмінності та вимоги до продукту в таблиці 4.5. Формування потреб потенційних клієнтів допоможе вибрати цільову аудиторію та розвивати проект в потрібному напрямку.

Таблиця 4.5 — Характеристики потенційних клієнтів стартап-проєкту

№	Потреба, що формує ринок	Цільова аудиторія	Відмінності у поведінці цільових груп клієнтів	Вимоги споживачів до товару
1	Автоматизоване тестування вебзастосунків	IT-компанії	Відсутні	Стабільність ПЗ
		IT-спільнота		Універсальність ПЗ
		Освітні заклади		Легкість розуміння ПЗ

Далі необхідно виконати аналіз ринкового середовища та визначити фактори загроз і можливостей.

Найпоширеніший спосіб боротьби із загрозами – це визначення можливих джерел виникнення загроз та створення плану реагування на них, обробку та мінімізацію руйнівних наслідків. Наведемо існуючі та можливі майбутні загрози та опишемо шляхи їх вирішення у таблиці 4.6.

Таблиця 4.6 — Фактори загроз

№	Фактор	Зміст загрози	Можлива реакція компанії
1	Перехід проєкту до автоматизованого тестування	Перехід проєкта до іншого підходу тестування займе певний час	Змушений крок для переводу проєкта на ефективний тип тестування
		Розробників необхідно навчити працювати з новим застосунком	
2	Оновлення проєкту	Необхідно адаптувати всі тестові сценарії до нової версії продукту	Кожне оновлення проєкту буде займати певний час для оновлення тестових сценаріїв
3	Відсутність сховища даних	Для перегляду та аналізу виконаних тестових сценаріїв необхідно мати достатній обсяг ресурсів	Підключення хмарних технологій для зберігання даних

За допомогою таблиці 4.7 розглянемо фактори можливостей, що матимуть позитивний вплив на розвиток стартап-проекту.

Таблиця 4.7 — Фактори можливостей

№	Фактор	Зміст можливості	Можлива реакція компанії
1	Низька конкуретність	На ринку відсутня конкурентність через малу кількість альтернативних рішень. Тож є велика ймовірність стати популярним продуктом	Удосконалення застосунку та впровадження нових технологічних рішень дозволить продукту стати популярним на ринку
2	Популярність сфери тестування	Сфера тестування на ринку ІТ набуває популярності, а ІТ-компанії бажають, щоб їх продукт був відмовостійкий від дефектів чи помилок	Компанія зацікавлена в наявності стартап-проєкту, оскільки зможе автоматизовано тестувати графічний інтерфейс користувача своїх продуктів

Продовження таблиці 4.7

3	Розвиток інструментальних засобів для автоматизації тестування	Надане архітектурне рішення в даному стартап-проєкті може набути подальшого розвитку, що призведе до потенційного розвитку інструментальних засобів для тестування	Розвиток займе певний час для удосконалення або розробки альтернативного ефективного рішення, проте успіх розвитку інструментального засобу неминучий

На основі проведеного аналізу можна зробити висновок, що суттєвою проблемою стартап-проєкту може стати: можлива нехватка ресурсів для детального звітування виконаного тесту, але ця проблема швидко вирішується при наявних доступних ресурсів в компанії. Оскільки для інших потенційних клієнтів не потребується стільки ресурсів порівняно з компанією. Серед можливостей є велика вірогідність зайняти нишу в сфері автоматизованого тестування графічного інтерфейсу користувача через низьку конкурентність та зростання популярності сфери тестування.

Розглянемо існуючі пропозиції на ринку та виділимо загальні риси характеристик, отриману інформацію описано в таблиці 4.8.

Таблиця 4.8 — Ступеневий аналіз конкуренції на ринку

№	Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства(можливі дії компанії, щоб бути конкурентоспроможною)
1	Тип конкуренції: олігополістична	На ринку наявна низька кількість альтернативних продуктів	Розробка універсального та надійного рішення, що дозволить закріпитися на вершині ринку
2	Рівень конкурентної боротьби: міжнародний	Альтернативні рішення розробляються по різним куточках світу	Вихід на міжнародний ринок ставить до продукту багато вимог, яким необхідно відповідати
3	Галузева ознака: внутрішньогалузева	Розроблений стартап-проєкт може використовуватися лише в IT-сфері	Наявність документації, зворотного зв’язку з користувачем та зручного графічного інтерфейсу вебзастосунка

Продовження таблиці 4.8

4	Конкуренція за видами товарів: товарно-видова	Боротьба на ринку між альтернативними продуктами	Універсальність продукту
5	Характер конкурентних переваг: нецінова	Наявність обширного функціоналу продукту привертає увагу більшість клієнтів	Удосконалювати та розвивати функціональність продукту
6	За інтенсивністю: не марочна	Рішення є новим на ринку ІТ та немає популярності	Проводити маркетингові дії для здобуття статусу марочної

На основі проведеного ступеневого аналізу конкуренції на ринку в ролі початкової стратегії можна обрати захоплення ринку шляхом міжнародною маркетинговою компанією для залучення користувачів та популяризації стартап-проєкта.

У таблиці 4.9 наведено аналіз умов конкуренції у галузі за методикою М.Портером.

Таблиця 4.9 — Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
	Lambda, Ranorex, Rapise	Невідомо	Набір функціоналу	Кожна IT компанія зацікавлена в тестуванні	Відсутні
Висновки	Існують	Дані відсутні	Чим більше функціоналу пропонує постачальник, тим краще рішення	Тестування має широку сферу застосування	Відсутні

Розглянемо фактори конкурентоспроможності, що наведені в таблиці 4.10.

Таблиця 4.10 — Обгрунтування факторів конкурентоспроможності

№	Фактор конкурентоспроможності	Обгрунтування
1	Автоматизація	Автоматизоване тестування заощаджує ресурси IT компанії, такі як: гроші, час, кількість годин праці тестувальника. Крім того, дозволяє швидко виконувати тестування та знаходити помилки завчасно

Продовження таблиці 4.10

2	Аналіз	Детальний звіт виконання тесту дає можливість проаналізувати виконаний тест. Кожний крок містить знімок екрану, що підтверджує виконання або невиконання кроку в тесті
3	Швидкодія	Швидкість виконання тесту надає можливості запускати їх більше, а аналіз проводити раніше
4	Підтримка	Впровадження нових функцій дій не повинна викликати помилки в інших місцях

За визначеними факторами конкурентоспроможності проведемо аналіз сильних та слабких сторін стартап-проекту, що наведено в таблиці 4.11.

Таблиця 4.11 — Порівняльний аналіз сильних та слабких сторін стартап-проєкту

№	Фактори конкурентоспроможності	Бали 1-20	Рейтинг товару-конкурентів в порівнянні з даним						
			-3	-2	-1	0	1	2	3
1	Автоматизація	10					+		
2	Аналіз	20						+	
3	Швидкодія	20				+			
4	Підтримка	10					+		

На фінальному етапі, на основі виділених ринкових загроз та можливостей, сильних та слабких сторін, складемо SWOT-аналіз стартап-проєкту, що наведено в таблиці 4.12.

Таблиця 4.12 — SWOT-аналіз стартап-проєкту

Сильні сторони (S):	Слабкі сторони (W):
Структуризація по проєктам створених тестів; Зберігання вхідних даних для тестів; Детальний графічний звіт виконання тесту; Універсальність функцій подій; Швидкість виконання тесту	Одночасний запуск 2-ух і більше тестів
Можливості (O):	Загрози (T):
Низька конкуретність; Популярність сфери тестування; Розвиток інструментальних засобів для автоматизації тестування	Перехід проєкту до автоматизованого тестування; Оновлення проєкту; Відсутність сховища даних

Враховуючи сильні сторони та можливості стартап-проєкту, наведені у матриці SWOT-аналізу. можна зробити висновок, що проєкт конкуретноспроможний на ринку.

4.4 Розроблення ринкової стратегії проєкту

Для розробки ринкової стратегії, потрібно визначити охоплення ринку за допомогою визначення груп потенційних споживачів. В таблиці 4.13 оглянути вибір цільових груп потенційних користувачів.

Таблиця 4.13 — Вибір цільових груп потенційних споживачів

№	Опис профілю цільової групи потенційних користувачів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	IT компанії, що розробляють вебзастосунки	Висока готовність	Високий попит	Середня конкуренція	Легко
2	IT спільнота, розробники або тестувальники для яких необхідно протестувати вебзастосунок	Висока готовність	Високий попит	Середня конкуренція	Легко
3	Освітні заклади, студенти чи школярі, котрі вивчають програмування та розробляють вебсайти	Низька готовність	Низький попит	Середня конкуренція	Легко

В якості цільових груп було обрано суспільства, котрі вивчають або впливають на ІТ. Оскільки стартап-проєкт позиціонує себе, як рішення для ІТ-суспільства.

В таблиці 4.14 описано базову стратегію розвитку стартап-проєкту.

Таблиця 4.14 — Визначення базової стратегії розвитку

Обрана альтернатива розвитку проєкту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
Розвиток в автоматизації тестування графічного інтерфейсу	Відкрите рішення, що надасть можливість розробникам пропонувати удосконалених або ефективних підходах в автоматизації тестування. Маркетингові підходи для залучення клієнтів	Швидкодія виконання тесту, впровадження штучного інтелекту та машинного навчання	Стратегія спеціалізації

В якості базової стратегії було обрано стратегію спеціалізацію, оскільки стартап-проєкт позиціонує себе, як рішення для ІТ-спільноти. Альтернативний

розвиток проєкту було обрано розвинення в автоматизації тестування графічного інтерфейсу

За допомогою таблиці 4.15 визначимо стратегію конкурентної поведінки на ринку.

Таблиця 4.15 — Визначення базової стратегії конкурентної поведінки

Чи є проєкт «першопрохідцем» на ринку	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, які?	Стратегія конкурентної поведінки
Ні	Стартап проєкт буде розвиватися, шукати нових споживачів та пропонувати кращі умови клієнтам конкурентів	Ні	Стратегія зайняття конкурентної ніші

На основі вимог споживачів обраних сегментів до стартап-проєкту та до продукту, в залежності від обраної базової стратегії розвитку та стратегії конкурентної поведінки, розробляється стратегія позиціонування, що полягає у формуванні ринкової позиції, за яким споживачі мають ідентифікувати торговельну марку/проєкт. В таблиці 4.16 визначено стратегію позиціонування.

Таблиця 4.16. — Визначення стратегії позиціонування

№	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проєкту	Вибір асоціацій, які мають сформувати комплексну позицію власного проєкту
1	Розуміння та зручність використання	Розроблення рішення, котре буде інтуїтивно зрозуміло будь-якому користувачу	Розроблені макети вебсторінок згідно правил вебдизайну	Привабливий дизайн, безпомилкова робота
2	Аналіз виконання тесту	Надати покроковий звіт виконання тесту з підтвердженням даних(знімок екрану, відео, логи)	Детальний аналіз кожного кроку тесту	Детальний звіт, зрозумілість, простота

Отже, цільовою групою користувачів є IT-компанії, що розробляють вебзастосунки, IT-спільнота, котра зацікавлена в тестуванні власних розроблених рішень та освітні заклади, студенти чи школяри яких вивчають програмування та розробляють вебсайти. Тобто весь ринок IT, тож базовою стратегію стартап-проєкта є стратегія спеціалізації. Оскільки на сьогоднішній день кількість користувачів в IT росте, тому за стратегію конкурентної поведінки було обрано стратегію заняття конкурентної ніші.

4.5 Розроблення маркетингової програми стартап-проєкту

Першим кроком до розроблення маркетингової програми стартап-проєкту є формування маркетингової концепції товару, який отримує споживач. Для цього у таблиці 4.17 підсумуємо результати попереднього аналізу конкурентоспроможності товару

Таблиця 4.17 — Визначення ключових переваг концепції потенційного товару

№	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1	Детальний звіт виконання тесту	Виконання тесту підкріплюється детальним звітом кожного кроку тестового сценарію	Як тільки виконується певний крок тестового сценарію про його статус відразу повідомляється в звіті
2	Додавання програмно нових функцій подій	Можливість внесення нових функцій дій, без змін в інших місцях ПЗ	Масштабованість застосунку та розширення потенційних функцій подій

Розглянемо трирівневу маркетингову модель продукту, наведену в таблиці 4.18, для уточнення ідеї продукту та його особливостей.

Таблиця 4.18 — Опис трьох рівнів моделі товару

<table>
<tr><th>Рівні товару</th><th colspan="3">Сутність та складові</th></tr>
<tr><td>1. Товар за задумом</td><td colspan="3">Програмне забезпечення для автоматизації тестування графічного інтерфейсу користувача</td></tr>
<tr><td rowspan="6">2. Товар у реальному виконанні</td><td>Властивості/ характеристики</td><td>М/Нм</td><td>Вр/Тх /Тл/Е/Ор</td></tr>
<tr><td>Якість</td><td>Нм</td><td>Тл</td></tr>
<tr><td>Масштабованість</td><td>М</td><td>Тл</td></tr>
<tr><td>Зрозумілість</td><td>М</td><td>Е</td></tr>
<tr><td colspan="3">Якість: перевірено тестуванням проєкту</td></tr>
<tr><td colspan="3">Пакування: відсутнє</td></tr>
<tr><td colspan="3">Марка: Testers</td></tr>
<tr><td>3. Товар із підкріпленням</td><td colspan="3">Ознайомлення клієнта з усіма сутностями та функціоналу проєкта для зрозумілого користування</td></tr>
</table>

Далі визначимо цінові меж, якими необхідно керуватись при встановленні ціни на потенційний товар, що передбачає аналіз ціни на товари-аналоги або товари субститути, а також аналіз рівня доходів цільової групи споживачів. Безпосередній аналіз проводиться експертним методом і представлено в таблиці 4.19.

Таблиця 4.19 — Визначення меж встановлення ціни

Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
0$	Щомісячні підписки від 15$ до 100$	IT компанії від 100 000$ за рік	Не визначена
		IT спільнота від 10 000$ за рік	
		Освітні заклади 0$ за рік	

Наступним кроком сформуємо систему збуту та опишемо у таблиці 4.20

Таблиця 4.20 — Формування системи збуту

Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
Цільові клієнти – IT компанії, які зацікавленні в впровадження автоматизованого тестування графічного інтерфейсу користувача в свої продукти	Постійний зворотній зв’язок. Удосконалення продукту. Покращення стартап проєкту згідно вимог клієнта	Від виробника до споживача	Прямий канал збуту

Останньою складової маркетингової програми є розроблення концепції маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів описано в таблиці 4.21.

Таблиця 4.21 — Концепція маркетингових комунікацій

№	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1	ІТ компанії, які зацікавленні в впровадження автоматизованого тестування графічного інтерфейсу користувача в свої продукти	Інтернет, електронна пошта	Позиція на основі автоматизованого тестування, швидкодії виконання тесту та легкість розуміння	Заохочення на тестування власних продуктів на коректну працездатність	Якісний продукт, це не тільки ідея, але й розробка й тестування

Продовження таблиці 4.21

2	IT спільнота, розробники чи тестувальники, котрі зацікавленні в тестуванні вебзастосунків	Інтернет, електронна пошта	Позиція на основі автоматизованого тестування, швидкодії виконання тесту та легкість розуміння	Заохочення на тестування власного вебзастосунку на коректність роботи	Будь впевненим в коректності роботи свого вебзастосунку. Протестуй та дізнайся
3	Освітні заклади, студенти чи школярі, котрі вивчають програмування та розробляють вебсайти	Інтернет, електронна пошта	Позиція на основі автоматизованого тестування, швидкодії виконання тесту та легкість розуміння	Заохочення протестувати своє рішення перед тим, як показати його	Протестуй декілька разів перед тим, як здати практичне завдання

Висновки до розділу

У даному розділі було описано зміст ідеї стартап-проєкту автоматизованого тестування графічного інтерфейсу користувача. Було проведено аналіз ринкових можливостей впровадження стартап-проєкту на IT-ринок. Після аналізу продукції конкурентів можна зробити висновок, що програмне забезпечення є конкурентоспроможним гравцем на ринку.

Для реалізації стартап-проєкту всі технології та можливість їх використання є доступними, тому проєкт реалізується без перешкод з технічної частини.

Під час аналізу ринкових можливостей запуску стартап-проєкту було виявлено, що попит на автоматизоване тестування зростає, тож кількість потенційних користувачів також збільшується.

На сьогоднішній день вимоги до сертифікації та стандартизації відсутні, що надає можливість впровадження стартап-проєкта на ринок без виконання додаткових вимог.

Проаналізувавши конкуренцію було виявлено, що конкуренти з різних країн, то рівень боротьби — міжнародний. Базова стратегія розвитку стартап-проєкту — стратегія спеціалізації, також було обрано альтернативний розвиток проєкту — розвиток в автоматизації тестування графічного інтерфейсу.

Було визначено маркетингову стратегію, для цього було проаналізовано вимоги від користувачів, канали комунікації, канали збуту та ключові переваги над конкурентами.

Провівши маркетинговий аналіз ринку, реалізація даного стартап-проєкту є доцільною.

ВИСНОВКИ

В даній магістерській роботі було розглянуто тему автоматизації тестування графічного інтерфейсу користувача. Було виконано дослідження даної предметної області, в якому було визначено методику та типи тестування, які необхідні для виконання поставлених задач. В огляді наукової літератури були розглянуті існуючі методи, їх переваги та недоліки. Тож розроблене рішення відповідає наступним критеріям:

- методика тестування: автоматизоване тестування;
- тип тестування: графічний інтерфейс користувача;
- метод тестування: створення тестових сценаріїв.

Проєктування програмного забезпечення розпочалося з розподілу по трьох незалежних модулів застосунку. Це модуль бази даних, модуль серверної частини та модуль клієнтської частини. Під час проєктування серверної частини було виявлено, що наявний трьохрівневий архітектурний підхід, не підходить для розробки програмного забезпечення згідно з вимогами. Оскільки впровадження нових функцій подій потребували змін на кожному рівні архітектури. Було запропоновано розробити новий рівень, котрий буде відповідати за функції подій. Перевага даного підходу полягає в при внесенні нових функцій подій нема потреби робити зміни в інших рівнях.

Реалізація програмного забезпечення відбувалася покроково. Перш за все було обрано технології, які доступні для реалізації поставлених задач. Під час реалізації бази даних було описано структуру та ER-діаграму. В реалізації серверної та клієнтської частини було описано використані підходи та розроблені класи.

Було проведено експериментальний дослід, котрий підтвердив ефективність розробленої архітектури. Порівняння відбувалося з трьох рівневою та чотирьох рівневою архітектурою серверної частини. Результати чотирьох рівневої в 5 разів краще по часу виконання всієї роботи та у 2 рази простіше для розробки.

Маркетинговий аналіз стартап-проєкта визначив рівень боротьби, стратегію, альтернативний розвиток розробленого рішення. Також згідно зі статистикою, сфера автоматизованого тестування набуває стрімкої популярності, а конкурентоспроможність розробленого рішення вважається високою.

Наукова новизна магістерської дисертації полягає в тому, що запропоновано архітектурне рішення за допомогою якого впровадження нових дій інструментального засобу автоматизованого тестування відбувається без необхідності змін в інших рівнях архітектури.

Практичне значення отриманих результатів полягає в тому, що розроблена архітектура допомагає легко впроваджувати універсальні функції подій для розробника програмного забезпечення. Кінцевий користувач отримує вебзастосунок в котрому має можливість створювати та запускати тестові сценарії.

Результати роботи магістерської дисертації опубліковано в одній статті.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Myers, Glenford J., Corey Sandler, and Tom Badgett. The art of software testing. John Wiley & Sons, 2011.
2) Garousi, Vahid, and Mika V. Mäntylä. "When and what to automate in software testing? A multi-vocal literature review." Information and Software Technology 2016. – 92-117c.
3) 7 Principles of Software Testing [Електронний ресурс] URL: https://www.functionize.com/blog/7-principles-of-software-testing
4) Thomas, Stephen W., et al. "Static test case prioritization using topic models." Empirical Software Engineering 2014. – 182-212c.
5) Raffelt, Harald, et al. "Dynamic testing via automata learning." International journal on software tools for technology transfer 2009. – 307-324c.
6) Coward, P. David. "A review of software testing." Information and Software Technology 1988. – 189-198c.
7) Metsa, Jani, Mika Katara, and Tommi Mikkonen. "Testing non-functional requirements with aspects: An industrial case study." Seventh International Conference on Quality Software (QSIC 2007). IEEE, 2007.
8) Beizer, Boris. Black-box testing: techniques for functional testing of software and systems. John Wiley & Sons, Inc., 1995.
9) Nidhra, Srinivas, and Jagruthi Dondeti. "Black box and white box testing techniques-a literature review." International Journal of Embedded Systems and Applications (IJESA) 2.2 2012. –: 29-50c.
10) Coulter, Andre' C. "Graybox software testing methodology: embedded software testing technique." Gateway to the New Millennium. 18th Digital Avionics Systems Conference. Proceedings (Cat. No. 99CH37033). Vol. 2. IEEE, 1999.
11) Mohammad, Sikender Mohsienuddin. "Automation Testing in Information Technology." International Journal of Creative Research Thoughts (IJCRT), ISSN 2015. – 2320-2882c.

12) Memon, Atif M., and Myra B. Cohen. "Automated testing of GUI applications: models, tools, and controlling flakiness." 2013 35th International Conference on Software Engineering (ICSE). IEEE, 2013.
13) Li, Kanglin, and Mengqi Wu. Effective GUI test automation: developing an automated GUI testing tool. Sybex, 2005
14) Stadie, Oliver, and Peter M. Kruse. "Closing gaps between capture and replay: Model-based gui testing." 1st INTUITEST Workshop. 2015.
15) M. Grechanik, Q. Xie and C. Fu, "Maintaining and evolving GUI-directed test scripts," 2009 IEEE 31st International Conference on Software Engineering, 2009. – 408-418c.
16) C. Fu, M. Grechanik and Q. Xie, "Inferring Types of References to GUI Objects in Test Scripts," 2009 International Conference on Software Testing Verification and Validation, 2009. – 1-10c.
17) E. Borjesson and R. Feldt, "Automated System Testing Using Visual GUI Testing Tools: A Comparative Study in Industry," *2012* IEEE Fifth International Conference on Software Testing, Verification and Validation, 2012. – 350-359c.
18) Wang, Fei, and Wencai Du. "A test automation framework based on web." 2012 IEEE/ACIS 11th International Conference on Computer and Information Science. IEEE, 2012.
19) Kurin, Erik, and Adam Melin. "Data-driven test automation: Augmenting gui testing in a web application.", 2013.
20) Danny R. Faught Keyword-Driven Testing [Електронний ресурс] URL: https://www.stickyminds.com/article/keyword-driven-testing
21) Laukkanen, Pekka. "Data-driven and keyword-driven test automation frameworks." Master's thesis. Helsinki University of Technology, 2006.
22) Tarun, Sharma, et al. "Keyword Based Testing of Windows Application."
23) PostgreSQL [Електронний ресурс] URL: https://www.postgresql.org/
24) Java [Електронний ресурс] URL: https://www.java.com/
25) Spring Framework [Електронний ресурс] URL: https://spring.io/
26) Maven [Електронний ресурс] URL: https://maven.apache.org/

27) Selenium [Електронний ресурс] URL: https://www.selenium.dev/

28) ChromeDriver [Електронний ресурс] URL: https://chromedriver.chromium.org/downloads

29) HTML [Електронний ресурс] URL: https://developer.mozilla.org/ru/docs/Web/HTML

30) LESS [Електронний ресурс] URL: https://lesscss.org/

31) Typescript [Електронний ресурс] URL: https://www.typescriptlang.org/

32) Angular [Електронний ресурс] URL: https://angular.io/

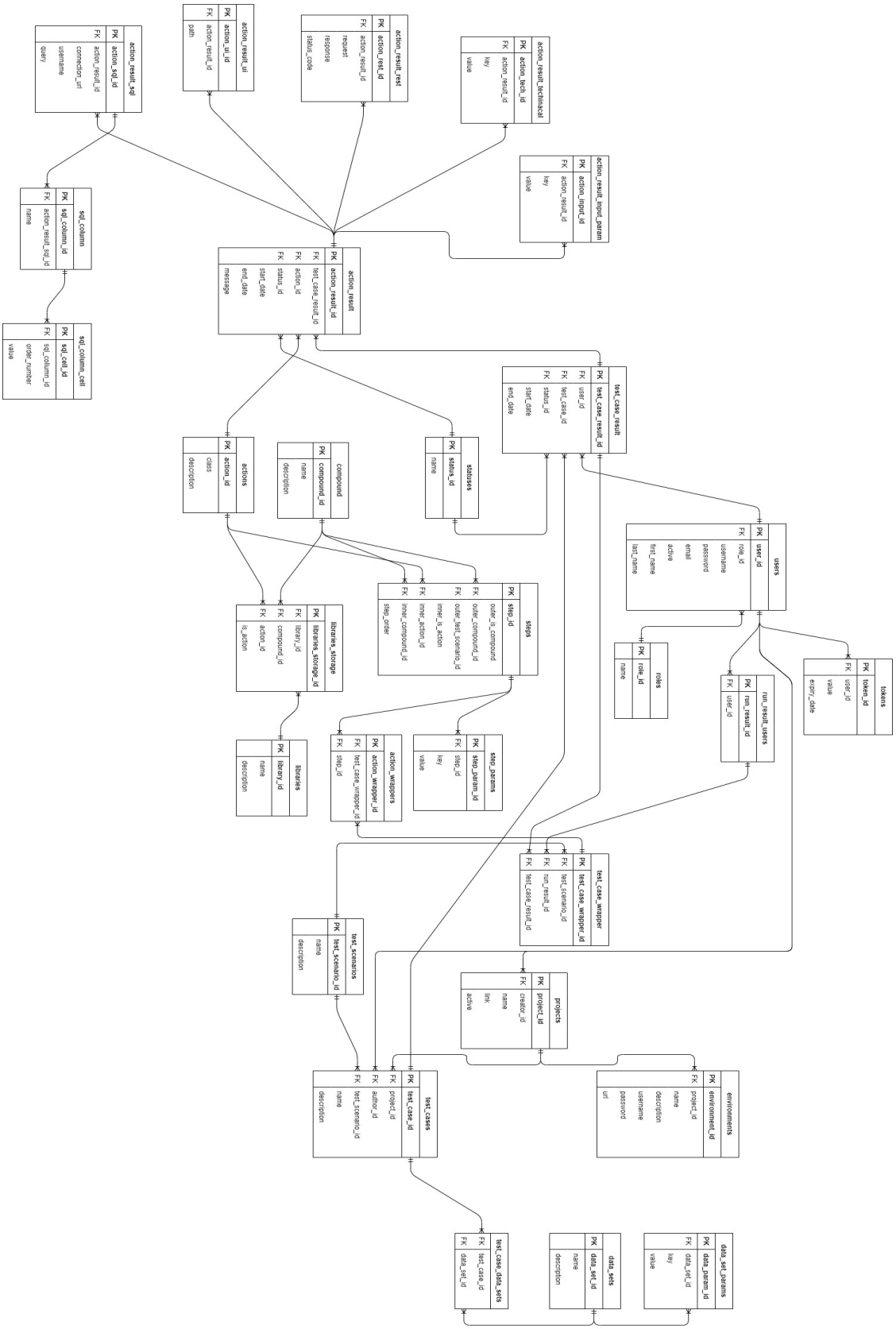
33) Angular Material [Електронний ресурс] URL: https://material.angular.io/

34) SockJS [Електронний ресурс] URL: https://www.npmjs.com/package/sockjs

ДОДАТКИ

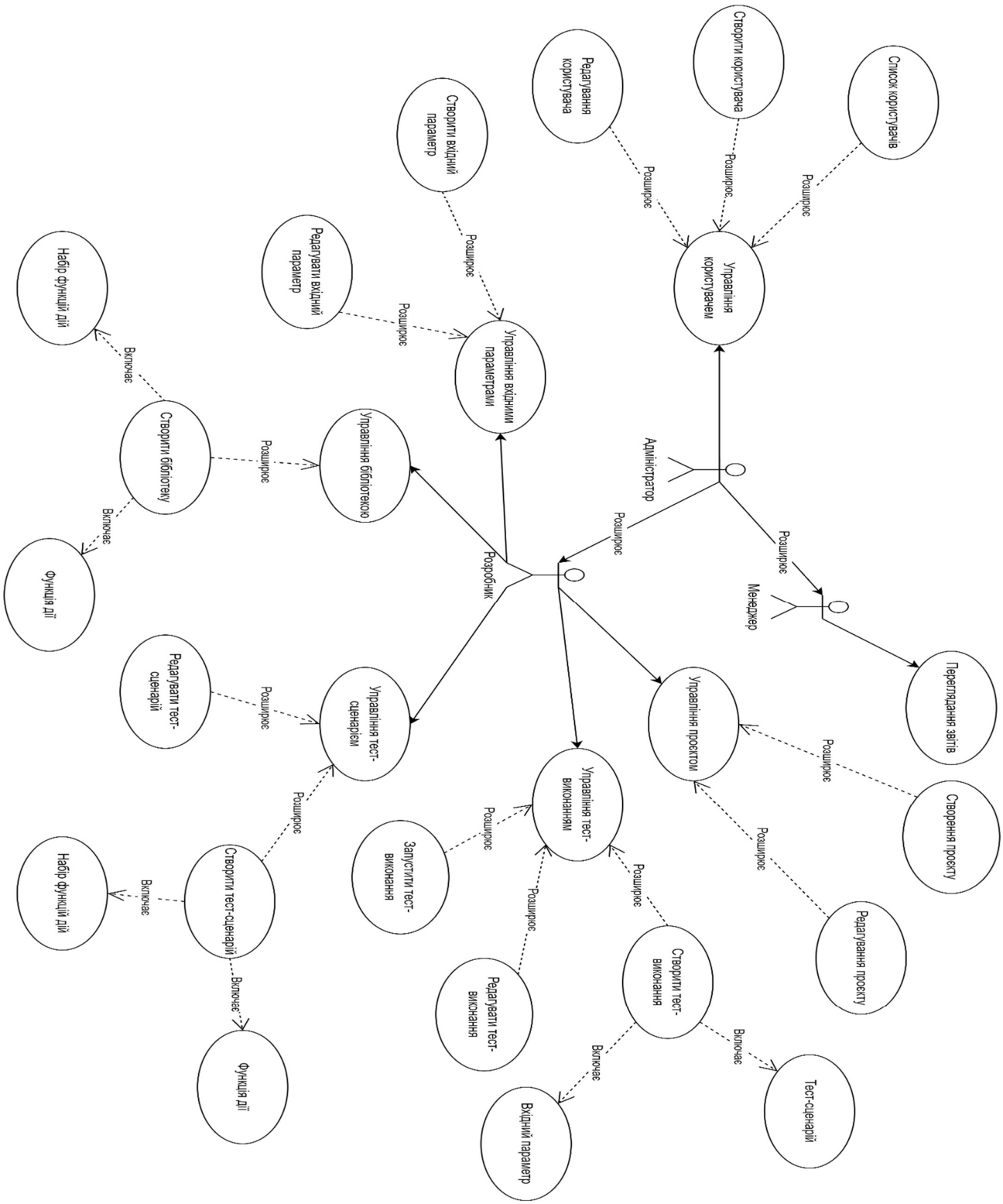
ДОДАТОК А

ER-діаграма бази даних



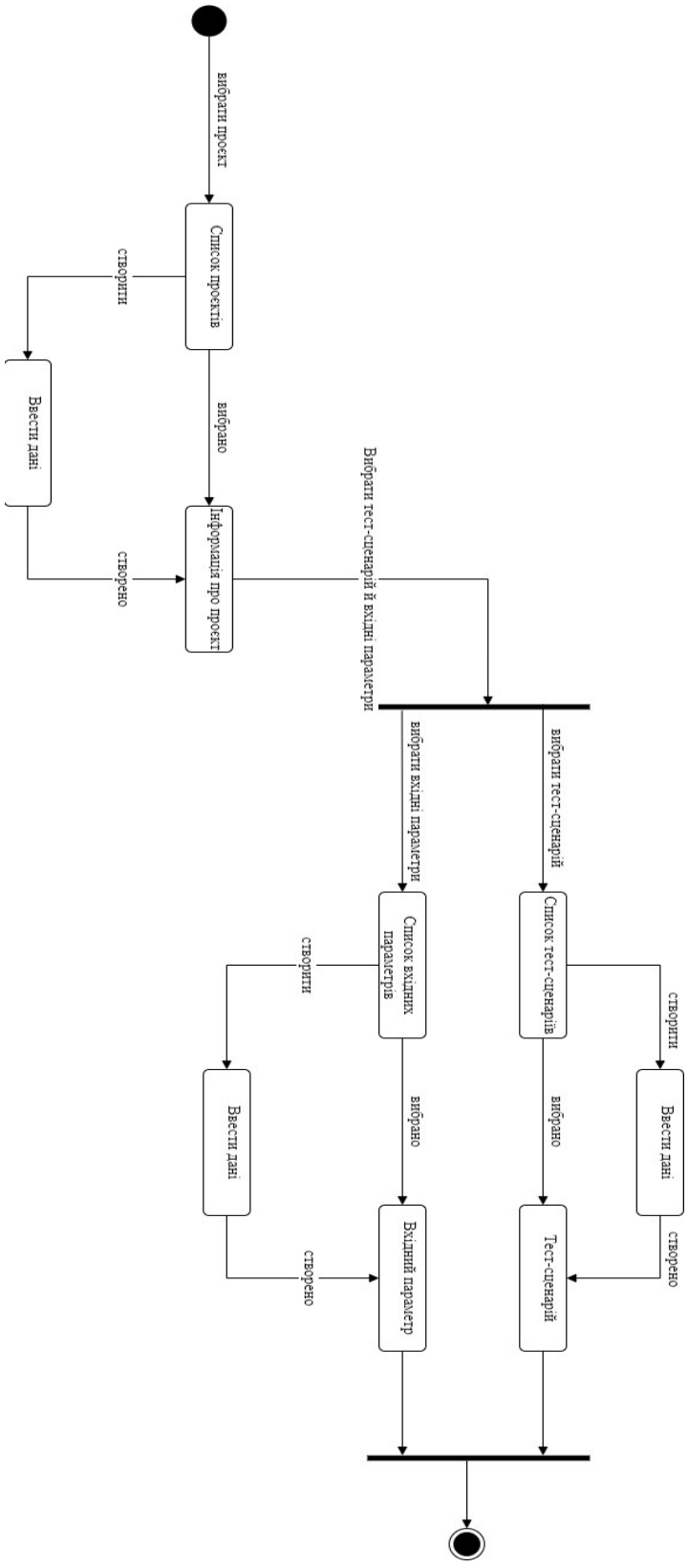
ДОДАТОК Б

Діаграма варіантів використань



ДОДАТОК В

Діаграма станів



ДОДАТОК Г

Лістинг коду

```
@FunctionEvent(
      name = "Click on element with ${xpath}",
      type = FunctionEventType.TECHNICAL,
      description = "Click on element with xpath",
      parameterNames = {"xpath"}
)
public class ClickOnElementWithXPath extends AbstractFunctionEvent {
  @Override
  protected FunctionEventResultTechnical logic(Map<String, String> params, WebDriver driver,
JdbcTemplate jdbcTemplate, Environment environment, RestTemplate restTemplate) {
      try {
         driver.findElement(By.xpath(params.get("xpath"))).click();
         return new FunctionEventResultTechnical();
      } catch (Exception e) {
         return new FunctionEventResultTechnical(new
FunctionEventExecutionException(e.getMessage()));
      }
  }
}

@FunctionEvent(
      name = "Click on element with ${id}",
      type = FunctionEventType.TECHNICAL,
      description = "Click on element with id",
      parameterNames = {"id"}
)
public class ClickOnElementWithId extends AbstractFunctionEvent {
  @Override
  protected FunctionEventResultTechnical logic(Map<String, String> params, Map<String, String>
context, WebDriver driver, JdbcTemplate jdbcTemplate, Environment environment, RestTemplate
restTemplate) {
      try {
         driver.findElement(By.id(params.get("id"))).click();
         return new FunctionEventResultTechnical();
      } catch (Exception e) {
         return new FunctionEventResultTechnical(new
FunctionEventExecutionException(e.getClass().getName()));
      }
  }
}

@FunctionEvent (
      name = "Set ${key} = label of element with ${xpath} in context",
      type = FunctionEventType.TECHNICAL,
      description = "Save element label to context",
      parameterNames = {"key", "xpath"}
)
public class SaveLabelByXPathFunctionEvent extends AbstractFunctionEvent {
```

```
   @Override
   protected FunctionEventResultTechnical logic(Map<String, String> params, Map<String,
String> context, WebDriver driver, JdbcTemplate jdbcTemplate, Environment environment,
RestTemplate restTemplate) {
      try {
         String text = driver.findElement(By.xpath(params.get("xpath"))).getText();
         context.put(params.get("key"), text);
         return new FunctionEventResultTechnical(Map.of("label", text));
      } catch (Exception e) {
         return new FunctionEventResultTechnical(new
FunctionEventExecutionException(e.getMessage()));
      }
   }
}
@FunctionEvent(
      name = "Take screenshot",
      type = FunctionEventType.UI,
      description = "Take screenshot"
)
public class ScreenshotFunctionEvent extends AbstractFunctionEvent {
   @Override
   protected FunctionEventResultUi logic(Map<String, String> params, Map<String, String>
context, WebDriver driver, JdbcTemplate jdbcTemplate, Environment environment, RestTemplate
restTemplate) {
      try {
         File screenshot = ((TakesScreenshot) driver).getScreenshotAs(OutputType.FILE);
         String filename = String.valueOf(new Date().getTime());
         String path = "~\\data_screen\\" + filename + ".png";
         File file = new File(path);
         FileUtils.copyFile(screenshot, file);
         return new FunctionEventResultUi(filename);
      } catch (Exception e) {
         return new FunctionEventResultUi(
               new FunctionEventExecutionException(e.getMessage()),
               "");
      }
   }
}
@FunctionEvent (
      name = "Execute ${query}",
      type = FunctionEventType.SQL,
      description = "Execute query",
      parameterNames = {"query"}
)
public class SqlFunctionEvent extends AbstractFunctionEvent {

   @Override
   protected FunctionEventResultSql logic(Map<String, String> params, Map<String, String>
context, WebDriver driver, JdbcTemplate jdbcTemplate, Environment environment, RestTemplate
restTemplate) {
      String query = params.get("query");
      try {
```

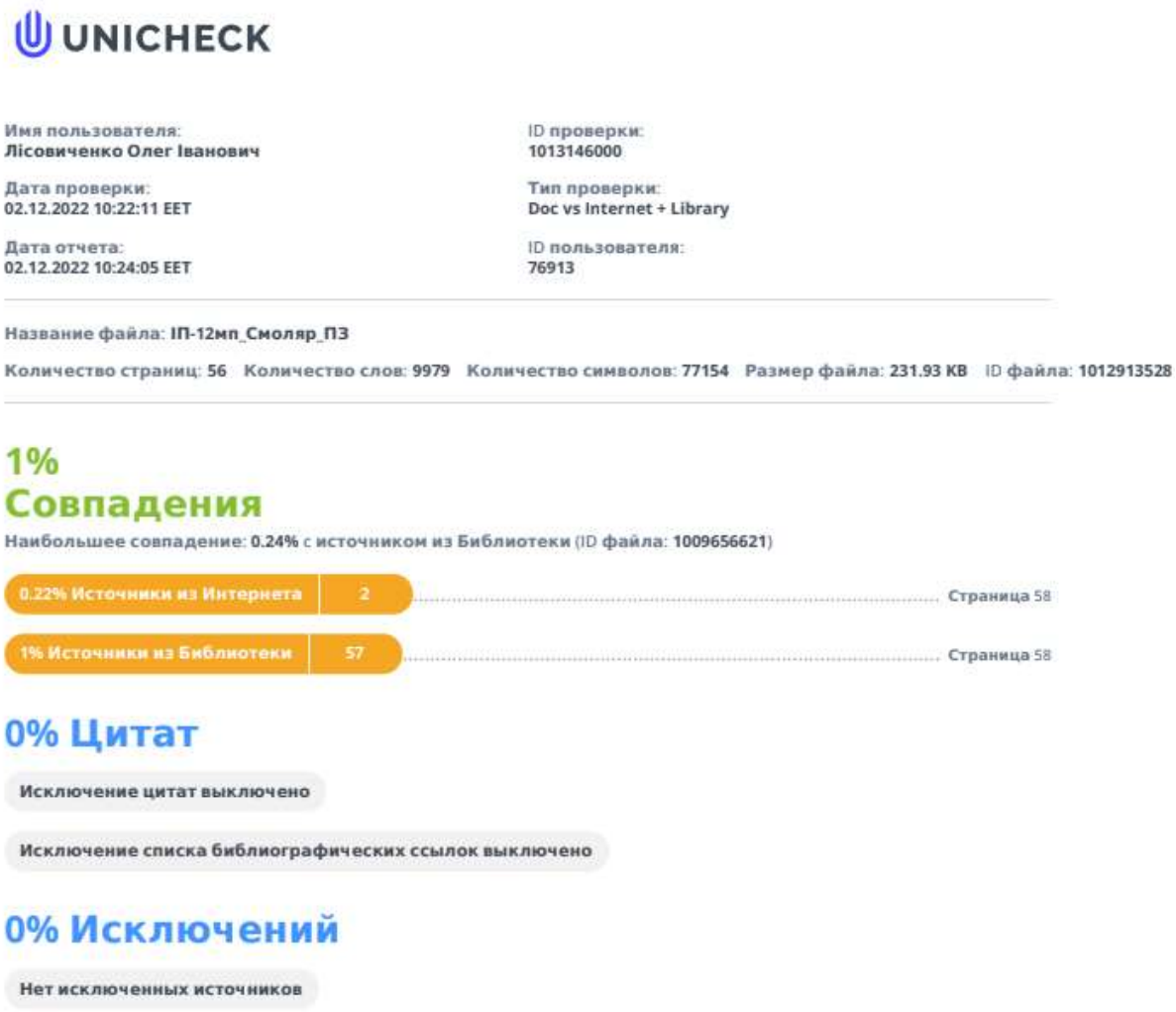
```
            List<SqlColumn> table = jdbcTemplate.query(
                    query,
                    SqlConverter::convertResultSetToTable);

            if (table != null) {
               return new FunctionEventResultSql(
                       environment.getUrl(),
                       environment.getUsername(),
                       query,
                       table);
            }

            throw new Exception("Query returned exception");
        } catch (Exception e) {
            return new FunctionEventResultSql(
                    new FunctionEventExecutionException(e.getMessage()),
                    environment.getUrl(),
                    environment.getUsername(),
                    query,
                    Collections.emptyList());
        }
    }
}
@FunctionEvent (
        name = "Wait ${n} seconds",
        type = FunctionEventType.TECHNICAL,
        description = "Wait some of time",
        parameterNames = {"n"}
)
public class WaitFunctionEvent extends AbstractFunctionEvent {
    @Override
    protected FunctionEventResultTechnical logic(Map<String, String> params, Map<String,
String> context, WebDriver driver, JdbcTemplate jdbcTemplate, Environment environment,
RestTemplate restTemplate) {
        try {
            Thread.sleep(Integer.parseInt(params.get("n")) * 1000);
            return new FunctionEventResultTechnical();
        } catch (Exception e) {
            return new FunctionEventResultTechnical(new
FunctionEventExecutionException(e.getMessage()));
        }
    }
}
@FunctionEvent (
        name = "Send ${method} request to ${url} with ${body}",
        type = FunctionEventType.REST,
        description = "Send request to url with body",
        parameterNames = {"method", "url", "body"}
)
public class UniversalRestFunctionEvent extends AbstractFunctionEvent {
```

```
    @Override
    protected FunctionEventResultRest logic(Map<String, String> params, Map<String, String>
context, WebDriver driver, JdbcTemplate jdbcTemplate, Environment environment, RestTemplate
restTemplate) {
        String requestBody = params.get("body");
        try {
            ResponseEntity<String> response = restTemplate.exchange(
                    params.get("url"),
                    HttpMethod.valueOf(params.get("method")),
                    new HttpEntity<>(requestBody),
                    String.class);
            return new FunctionEventResultRest(
                    requestBody,
                    response.getBody() != null ? response.getBody() : "",
                    response.getStatusCodeValue());

        } catch (Exception e) {
            return new FunctionEventResultRest(
                    new FunctionEventExecutionException(e.getMessage()),
                    requestBody,
                    "",
                    0);
        }
    }
}
@FunctionEvent (
        name = "Send get request to ${url}",
        type = FunctionEventType.REST,
        description = "Send get request to url",
        parameterNames = {"url"}
)
public class GetRestFunctionEvent extends AbstractFunctionEvent {

    @Override
    protected FunctionEventResultRest logic(Map<String, String> params, Map<String, String>
context, WebDriver driver, JdbcTemplate jdbcTemplate, Environment environment, RestTemplate
restTemplate) {
        try {
            ResponseEntity<String> response = restTemplate.exchange(
                    params.get("url"),
                    HttpMethod.GET,
                    null,
                    String.class);
            return new FunctionEventResultRest(
                    "",
                    response.getBody() != null ? response.getBody() : "",
                    response.getStatusCodeValue());

        } catch (Exception e) {
            return new FunctionEventResultRest (
                    new FunctionEventExecutionException(e.getMessage()),
                    "",
```

```
                "",
                0);
        }
    }
}
@FunctionEvent (
        name = "Input ${text} into field with ${id}",
        type = FunctionEventType.TECHNICAL,
        description = "Input provided text into field with id",
        parameterNames = {"text", "id"}
)
public class InputTextIntoFieldWithIdFunctionEvent extends AbstractFunctionEvent {
    @Override
    protected FunctionEventResultTechnical logic(Map<String, String> params, Map<String,
String> context, WebDriver driver, JdbcTemplate jdbcTemplate, Environment environment,
RestTemplate restTemplate) {
        try {
            driver.findElement(By.id(params.get("id"))).sendKeys(params.get("text"));
            return new FunctionEventResultTechnical();
        } catch (Exception e) {
            return new FunctionEventResultTechnical (new
FunctionEventExecutionException(e.getClass().getName()));
        }
    }
}
```

ДОДАТОК Д

Результати перевірки роботи на співпадіння



UNICHECK

Имя пользователя:
Лісовиченко Олег Іванович

ID проверки:
1013146000

Дата проверки:
02.12.2022 10:22:11 EET

Тип проверки:
Doc vs Internet + Library

Дата отчета:
02.12.2022 10:24:05 EET

ID пользователя:
76913

Название файла: ІП-12мп_Смоляр_ПЗ

Количество страниц: 56 Количество слов: 9979 Количество символов: 77154 Размер файла: 231.93 KB ID файла: 1012913528

1%
Совпадения

Наибольшее совпадение: 0.24% с источником из Библиотеки (ID файла: 1009656621)

0.22% Источники из Интернета 2 ..... Страница 58

1% Источники из Библиотеки 57 ..... Страница 58

0% Цитат

Исключение цитат выключено

Исключение списка библиографических ссылок выключено

0% Исключений

Нет исключенных источников