ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ СЕМЕНА КУЗНЕЦЯ

ФАКУЛЬТЕТ ЕКОНОМІЧНОЇ ІНФОРМАТИКИ

КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ

Пояснювальна записка

до дипломного проєкту

бакалавра

на тему: «Розробка фреймворку для автоматизації тестування інтерфейсу веб-сайту»

Виконав: студент 4 курсу,
групи 6.04.51.16.01,
спеціальності 122
«Комп'ютерні науки
та інформаційні технології»
Гаркавий С. О.

Керівник: к.е.н., доц.
Беседовський О. М.

Харків – 2020 рік

РЕФЕРАТ

Пояснювальна записка до дипломного проекту: 73 с., 15 рис., 27 табл., 1 додаток, 34 джерела.

Об'єктами проектування є функціональні елементи, архітектура, інформаційне і програмне забезпечення фреймворку автоматизованого тестування інтернет-магазину.

Мета проектування – створення фреймворку автоматизованого тестування, що забезпечує зручність під час створення автоматизованих тестів для широкого кола спеціалістів-тестувальників.

Метод проектування – використання програмних систем ARIS, Ramus Educational, ERWin, Intellij IDEA, онлайн-сервісу Draw.io.

Створений фреймворк дозволяє забезпечити виконання тестування інтерфейсу користувача без участі тестувальника. Використання фреймворку автоматизованого тестування дозволяє скоротити затрати людських ресурсів на різних етапах тестування програмного забезпечення і підвищити ефективність процесу тестування інтерфейсу веб-сайту.

Процес автоматизації тестування є відносно новим напрямком в процесі життєвого циклу розробки програмного забезпечення. Проте вже зараз все більше підприємств використовують автоматизовані фреймворки на своїх проектах, оцінивши підвищення швидкості та якості процесу тестування.

Результати розробки можуть бути впроваджені на ІТ-підприємствах, а саме в командах, що займаються розробкою та тестуванням веб-сайтів.

ФРЕЙМВОРК АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ, ТЕСТОВИЙ СЦЕНАРІЙ, ТЕСТОВИЙ НАБІР, ВІДСОТОК ПРОХОДЖЕННЯ ТЕСТІВ, ГРАФІЧНИЙ ІНТЕРФЕЙС, ВЕБ-САЙТ, ШАБЛОН PAGE OBJECT, SELENIDE, ЗВІТ ALLURE.

ABSTRACT

The bachelor’s thesis report: 73 pages, 15 figures, 27 tables, 1 appendices, 34 sources.

The objects of designing are functional elements, architecture and software of test automation framework for internet-shop.

The purpose of designing is to create test automation framework, which assures usability during creation of automated test by wide range of QA engineers.

The method of designing is using the ARIS, ramus Educational, ERWin, Intellij IDEA, Draw.io online service.

Designed framework allows to assure GUI testing without QA specialists’ intervention. Usage of test automation framework allows to reduce human resource costs during different stages of QA process and increase its efficiency.

Test automation process is relatively new flow in software development lifecycle. There are many enterprises, that use test automation frameworks in their projects because of speed and quality increase in QA process.

The obtained results can be applied at IT-enterprises. Especially it can be used by teams, that take part in web development.

TEST AUTOMATION FRAMEWORK, TEST CASE, TEST SUIT, PASS RATE, GUI, WEB SITE, PAGE OBJECT PATTERN, SELENIDE, ALLURE REPORT.

ЗМІСТ

ВСТУП

Контроль якості програмного забезпечення – планомірна і систематична програма дій, що покликана гарантувати відповідність системи бажаним характеристикам.

Тестування програмного забезпечення являється важливою частиною процесу розробки. Завдяки йому є можливість перевірити та оцінити відповідність вимог до програмного продукту та реального продукту. Проте по мірі того, як програмний продукт імплементує все більше функціональності, тестування класичними методиками стає все більш затратним і все менш ефективним.

Мета автоматизації тестування – автоматичне виконання тестів замість ручної праці спеціалістів-тестувальників. Автоматизація тестування є актуальним питанням для підприємств в сфері розробки програмного забезпечення, оскільки дозволяє скоротити затрати часу і необхідних спеціалістів для організації процесу тестування.

Проте на даний момент є проблема залучення спеціалістів ручного тестування в сферу підтримки автоматизованих тестів. Для цього створюються спеціальні допоміжні інструменти – фреймворки автоматизованого тестування, що дають можливість зосередитись на виконанні конкретної задачі без розробки допоміжного та службового інструментарію.

Метою даного проекту є створення універсального фреймворку для автоматизації процесу тестування інтерфейсу веб-додатків. Предметна область проектування – процес розробки програмного забезпечення, об’єктом проектування є забезпечення якості та надійності програмного продукту.

В процесі розробки було використано ряд спеціалізованих інструментів, бібліотек та технологій. Для забезпечення роботи тестових сценаріїв було використано інструмент для тестування TestNG, для роботи з веб-сторінками використовувались засоби фреймворку Selenide. Збереження даних про тестування відбувається за допомогою СКБД MySQL, а генерація звіту про останній запуск тестового набору покладена на засіб Allure Report.

Практична значимість даного проекту полягає в тому, що даний фреймворк дозволяє з легкістю провадити процес автоматизованого тестування будь-якого веб-сайту. Окрім того невисокий поріг входження полегшує взаємодію користувача з фреймворком і тому може використовуватись як індивідуальними розробниками програмного забезпечення, так і командами тестувальників на IT підприємствах.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ «ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ»

1.1 Коротка характеристика об’єкта управління «EPAM Systems»

1.1.1 Опис напрямків діяльності об’єкта управління

EPAM Systems – один із найбільших виробників програмного забезпечення в Східній Європі. Компанія займається розробкою проектного (замовного) програмного забезпечення, а також лишається одним із провідних гравців в сфері консалтингу в Центральній та Східній Європі.

Маючи в своєму складі понад 5000 співробітників, компанія має представництва в більш ніж 30 країнах світу. Офіси компанії розташовані в таких країнах як Білорусь, Росія, Україна, Казахстан, Сполучені Штати Америки, Угорщина, Великобританія, Німеччина, Швеція та Швейцарія.

Серед основних напрямків діяльності компанії можна виділити два основних: обслуговування програмного забезпечення та бізнес-додатків і IT-консалтинг з урахуванням галузевої специфіки бізнесу.

В першому випадку можна розбити напрямок діяльності на більш специфічні. Серед них необхідно виділити такі етапи: розробка, тестування, супровід та підтримка відповідного програмного продукту чи бізнес-додатку.

1.1.2 Опис організаційної структури підприємства

В IT-компаніях найпоширенішими є три види організаційної структури: матрична, лінійна та проектна. Лінійна структура являється найпоширенішою і в основі її полягає існування невеликих крос-функціональних команд (відділів в класичній термінології). Дана структура має значні переваги, в першу чергу завдяки стійкості в кризових ситуаціях та швидкій ескалації проблем. Проте часто вирішення проблем підрозділу знаходиться за рамками повноважень керівника (наприклад, нестача людських ресурсів).

Проектна структура дуже подібна до лінійної. Вона також складається з невеликих крос-функціональних команд, в яких повноту влади має проектний менеджер або lead. Основна різниця полягає в тому, що при даному варіанті структури співробітники наймаються виключно під конкретний проект і по його закінченню робота з ними припиняється. Основна перевага такої структури полягає у відсутності необхідності перерозподілу персоналу між іншими проектами після закінчення поточного. Натомість недоліком можна вважати фактичну відсутність обміну досвідом та взаємодію між проектами

Матрична структура вносить поділ в функції влади. Проектні менеджери виділяються в окремий відділ, а функції планування, організації та контролю

поділяються між керівництвом проекту та керівниками функціональних напрямків (керівниками відділу розробки, тестування тощо). Дана структура є найважчою до впровадження, адже при наявності невеликих проектів можливі ускладнення комунікації між відділами і як наслідок спад продуктивності. Проте у випадку вдалої імплементації матрична структура сприяє накопичуванню знань в рамках кожного окремого функціонального відділу, що сприяє підвищенню ефективності праці.

В “EPAM Systems” використовується лінійна структура організації. Більш детальна інформація про структурну організацію компанії не може бути надана, оскільки є комерційною таємницею, розголошення якої буде порушенням договору про нерозголошення. Тому далі буде представлена схематично лінійна організаційна структура.

Схема наведена на рис. 1.1.

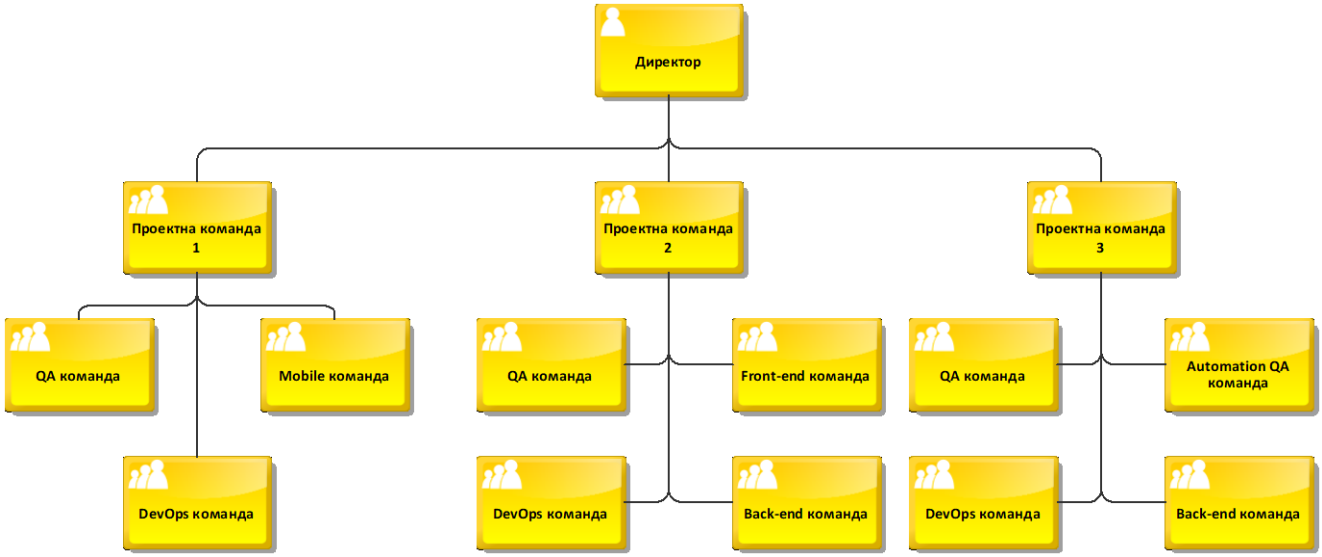

Рисунок 1.1 – Схема лінійної структури

1.1.3 Визначення проблем автоматизації тестування

Відповідно до стандарту IEEE SWEBOOK v3.0 «тестування програмного забезпечення» (Software testing) – це перевірка відповідності між реальною та очікуваною поведінкою програмного продукту, здійснюваного на кінцевому наборі тестів, обраним певним чином[1].

Процес тестування програмного забезпечення є затратним процесом, адже являється основний інструментом забезпечення якості програмного продукту. Відповідно до довідника стандартизації та сертифікації програмного забезпечення

всі види тестування, відповідно до мети проведення, можна умовно розділити на функціональне, нефункціональне та тестування пов’язане зі змінами[2].

Рисунок 1.2 – Схема стандартного складу Automation QA команди

Функціональні тести базуються на функціях та особливостях системи, а також взаємодії з іншими системи. Тому функціональні тести можуть бути представлені на будь-якому рівні тестування: модульному, інтеграційному чи системному.

Нефункціональне тестування описує тести, необхідні для визначення характеристик програмного забезпечення, які можуть бути виміряні певними величинами. Найкращим прикладом нефункціонального тестування буде тестування продуктивності.

Таким чином, стає помітно, що тестування охоплює майже всі аспекти діяльності програмного продукту. Зазвичай кількість тестів та тестових наборів є надто великою для обробки працівником. В такому випадку має сенс автоматизація тестування.

Автоматизація тестування («Software Automation Testing»)[6] – це процес верифікації програмного забезпечення, під час якого основні функції і кроки тестування виконуються автоматично за допомогою інструментів автоматизованого тестування.

Основною проблемою під час автоматизації тестування являється підготовка та контроль над тестами та їх наборами, а також налаштування необхідного оточення для запуску та виконання автоматичних тестів. При цьому спостерігається проблема з залученням спеціалістів, які не мають специфічних знань мов програмування та засобів розробки.

Для написання автоматизованих тестів більшість засобів вимагають від тестувальника навичок роботи зі скриптовими мовами такими як VB Script, JavaScript, тощо. Зазвичай такі інструменти дозволяють створювати тести за допомогою запису та відтворення певного набору дій в браузері, проте такі скрипти важко повторно використати та підтримувати.

Фреймворк автоматизованого тестування – це набір концепцій, умов та практик, направлених на повторне використання коду, зменшення затрат на підтримку та підняття надійності тестів.

Використання фреймворку автоматизованого тестування здатне вирішити проблеми повторного використання та стабільності тестів та тестових наборів, а також не потребує спеціальних знань, що сприяє використанню фреймворка широким колом спеціалістів (зокрема розробників, спеціалістів з ручного тестування та представників бізнесу).

1.2 Опис предметної області «Автоматизація тестування»

Процес тестування як було зазначено в параграфі 1.1.3, є процесом всебічним і об’ємним. На даному етапі не існує чіткого та єдино правильного алгоритму роботи цього процесу. Багато команд впроваджують Agile-практики в процес тестування, що робить даний процес більш ітеративним ніж послідовним. Проте під час кожної ітерації виконується відносно сталий набір функцій. Діаграма IDEF0 для процесу тестування наведена на рис. 1.3. Декомпозиція діаграми IDEF0 наведена на рис. 1.4

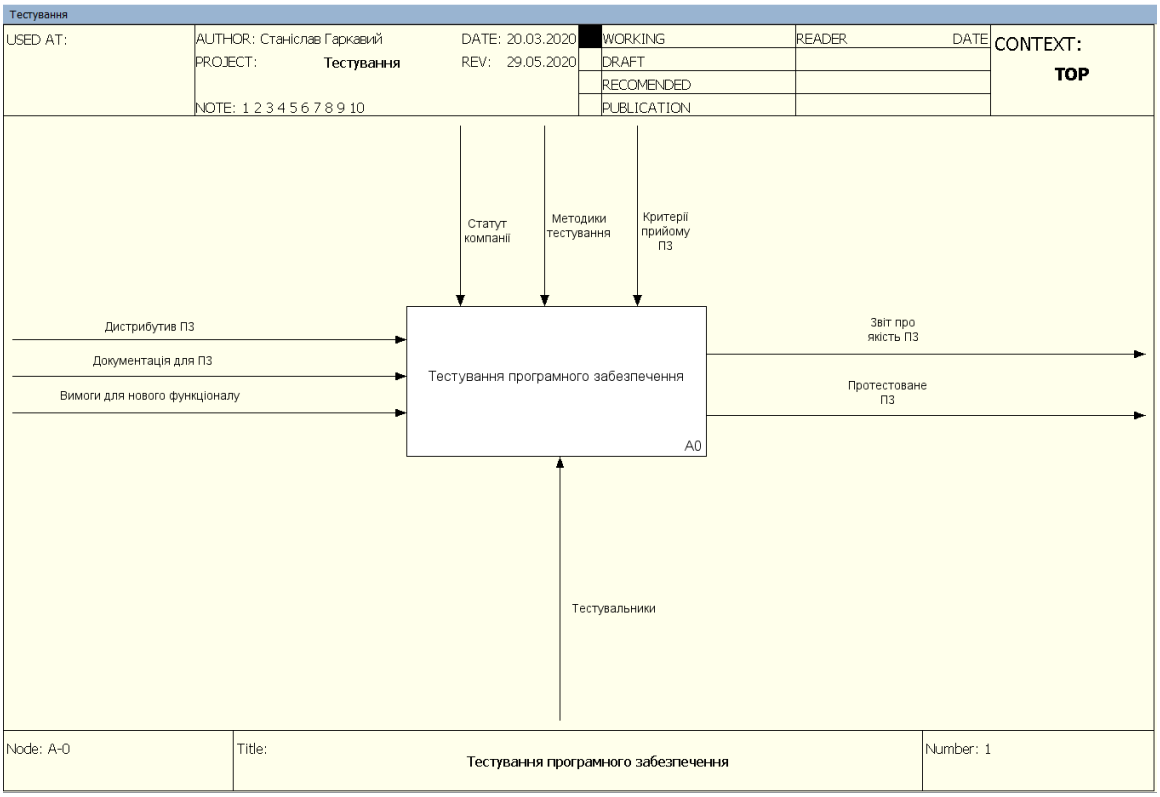

Рисунок 1.3 – Діаграма IDEF0 процесу «Тестування ПЗ»

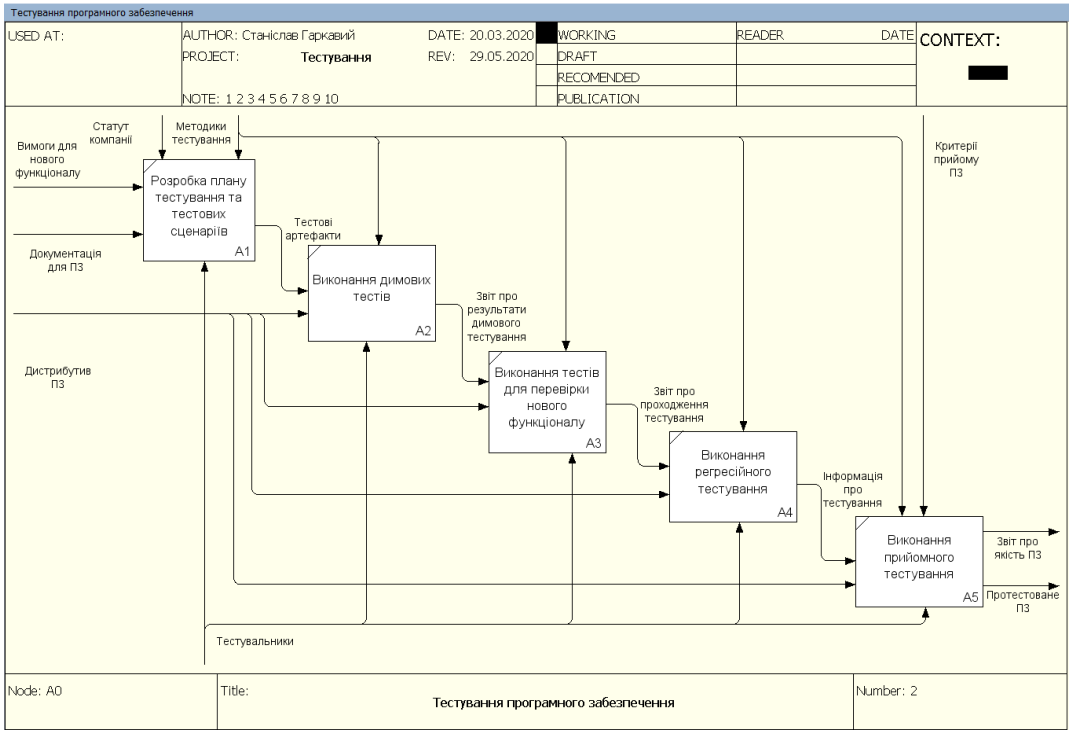

Рисунок 1.4 – Декомпозиція діаграми IDEF0

Як можна спостерігати на рис. 1.3, на вхід даного процесу подається актуальний дистрибутив програмного продукту (зазвичай в системі CI/CD), документація для продукту, а також вимоги, які мають бути протестовані.

Серед функцій керування для цього процесу варто відмітити Статут IT-компанії, методики тестування програмного забезпечення та критерії прийому програмного забезпечення. Механізмами ж даного процесу можна вважати тестувальників, lead команди тестувальників, замовника та розробників.

На виході даного процесу є інформація про якість програмного продукту, статистика тестування, а також протестована версія актуального програмного продукту.

На декомпозиції (рис. 1.4) можна побачити, що даний процес має в собі 5 етапів: створення плану тестування та розробка тестів, димове тестування, тестування нового функціоналу, регресійне тестування та прийомне тестування.

На етапі планування lead тестувальників і тестувальники отримують документацію та вимоги до програмного продукту. На основі цього відбувається планування та розробка тестових сценаріїв, які далі будуть проведені та впроваджені в фреймворку автоматизованого тестування.

На етапі димового тестування відбувається перевірка тестів, що відповідають за основну функціональність програмного продукту. Для цього повторно виконуються вже існуючі димові тести.

На етапі тестування нового функціоналу буде проведено тестування відповідно до тестових сценаріїв, розроблених на першому етапі.

На етапі регресійного тестування командою тестувальників буде повторено димові тести, виконані на другому етапі. Метою цього є перевірка відсутності дефектів основного функціоналу програми, пов’язаних з впровадженням нового функціоналу. Після виконання цього етапу формується статистика тестування, в якій зберігаються необхідні метрики тестування (кількість тестів тощо).

На етапі прийомного тестування відбувається повне тестування програмного продукту за участю представників з боку замовника та відповідно до критеріїв прийому програмного продукту. Після цього формується звіт про тестування, в якому зазначаються такі метрики, як кількість тестових сценаріїв, відсоток проходження тестів, кількість знайдених дефектів програмного продукту, тощо.

Характеристика бізнес-процесу наведена у табл. 1.1.

Таблиця 1.1 – Характеристика бізнес-процесу

Назва характеристики	Значення характеристики
1	2

Закінчення таблиці 1.1

1	2
Ім’я бізнес-процесу	Тестування програмного продукту
Основні учасники	Lead QA команди, роль: планування та контроль робочого процесу Тестувальники QA команди, роль: проходження тестових сценаріїв та збір статистичної інформації Замовник, роль: затвердження відповідності програмного продукту вимогам Розробник: підтримка тестувальників та хотфікс знайдених дефектів
Вхідна подія	Поява в системі CI/CD нової версії ПЗ
Вхідні документи	Документація програмного продукту, вимоги, дистрибутив програмного продукту
Вихідна подія	Реліз нової версії ПЗ
Вихідні документи	Статистика тестування, дані про якість ПЗ та протестована версія ПЗ
Клієнт бізнес-процесу	Команда розробників, замовник

1.3 Огляд і аналіз наявних аналогів, що реалізують функції предметної області

Ринок фреймворків для автоматизації тестування доволі широкий і росте з кожним роком. Виробники намагаються покращити свої програмні продукти для більш зручного процесу автоматизації та розширити можливості застосування своїх фреймворків.

Проте часто буває, що в погоні за функціональністю продукт втрачає свої переваги. Це може бути як втрата зручності користування, так і висока ціна фреймворку. Значний обсяг можливостей для автоматизації може вимагати значних потужностей від обладнання, на якому запускаються тести.

Іншим боком проблеми є відсутність будь-якої підтримки фреймворку з боку виробника. В такому випадку є позитивним фактором відкритість вихідного програмного коду, яка дозволяє робити надбудови над основними бібліотеками самими користувачами.

Важливим питанням є й питання сумісності. Фреймворк може використовувати такі технічні рішення, які не дозволяють його використання на певній операційній системі чи браузері.

З точки зору бізнесу важливим питанням є складність знаходження спеціалістів, які мають навики написання та підтримки тестів з використанням даного фреймворку. Для вирішення цієї проблеми варто оцінити два фактори: мова програмування на якому написано фреймворк і підтримка специфічних моделей тестування.

Від мови програмування на якій написано фреймворк в першу чергу залежить наскільки легко знайти спеціаліста зі знанням даної мови. Натомість підтримка специфічних моделей тестування дозволяє спростити розуміння коду для людей, які не мають спеціалізованих знань.

Найважливішими моделями тестування можна назвати BDD (Behaviour Driven Development) і KDD (Keyword Driven Development). Будучи близькими між собою, обидва підходи декларують використання опису поведінки тесту в першому випадку та використання ключових слів в другому. Незалежно від обраної моделі автоматизований тест пишеться нативною мовою і може бути запущений користувачем без специфічних знань [12].

З технічної точки зору важливим пунктом також є підтримка DDD (Data Driven Development). Дана модель полягає в об’єднанні схожих тестів в один на вхід, якому подаватимуться різні набори даних. Таким чином використання ресурсу тестувальника стає більш продуктивним, адже зменшується час автоматизації та підвищується відсоток повторного використання коду[8].

Останнім, але не за значимістю є підтримка сторонніх бібліотек в рамках Continuous Integration/Delivery. Від підтримки цих бібліотек залежить наскільки легко вбудувати тестування в процес постійної розробки та поставки ПЗ. Це в свою чергу впливає на швидкість впровадження нових тестів, а також отримання актуальної версії програмного продукту.

З урахуванням вищеназваних критеріїв було проведено огляд наявних продуктів на ринку фреймворків автоматизованого тестування. Найбільш підходящими були обрані наступні програмні продукти: Selenium, Katalon, UFT, Test Complete, Watir [24].

Порівняльна характеристика основних фреймворків автоматизації тестування наведено на табл. 1.2.

Таблиця 1.2 – Характеристика основних фреймворків

Категорія	Selenium	Katalon	UFT	Test Complete	Watir
1	2	3	4	5	6
Підтримка	Так	Так	Так	Так	Так
Ціна	Безкоштовно	Безкоштовно	800 USD	1000 USD	Безкоштовно
Відкритість	Так	Ні	Ні	Ні	Так
Підтримка браузерів	IE, Chrome, Firefox, Opera, Edge, Мобільні браузери	IE, Chrome, Firefox	IE, Chrome, Firefox, Safari, Мобільні браузери	IE, Chrome, Firefox, Opera, Safari, Edge	IE, Chrome, Firefox, Opera, Safari, Edge
Підтримка ОС	Windows, Linux, OS X	Windows, Linux, OS X	Windows	Windows	Windows, Linux, OS X
Мова програмування	Java, C#, Perl, Python, JavaScript, Ruby, PHP	Java/Groovy	VBScript	JavaScript, Python, VBScript, Jscript, Delphi, C++, C#	Ruby
CI/CD	Так	Так	Так	Так	Так
BDD, KDD, DDD	Ні	Ні	Ні	Так	Так

Selenium – один із найпопулярніших фреймворків для автоматизації тестування. Написаний на мові Java, він здатен знаходити та розпізнавати в файлах команди для автоматизованого управління браузером. Фактично його принцип роботи полягає у виконанні набору скриптів, заданих в певному файлі.

Окрім своєї простоти і надійності, Selenium став популярним завдяки підтримці різних операційних систем, різних мов програмування та різних браузерів. Проте основним його недоліком є високий поріг входження, адже

надаючи великий інструментарій, Selenium змушує користувача витрачати час на самостійну конфігурацію та створення бібліотек[21].

Приклад графічного інтерфейсу наведено на рис. 1.5.

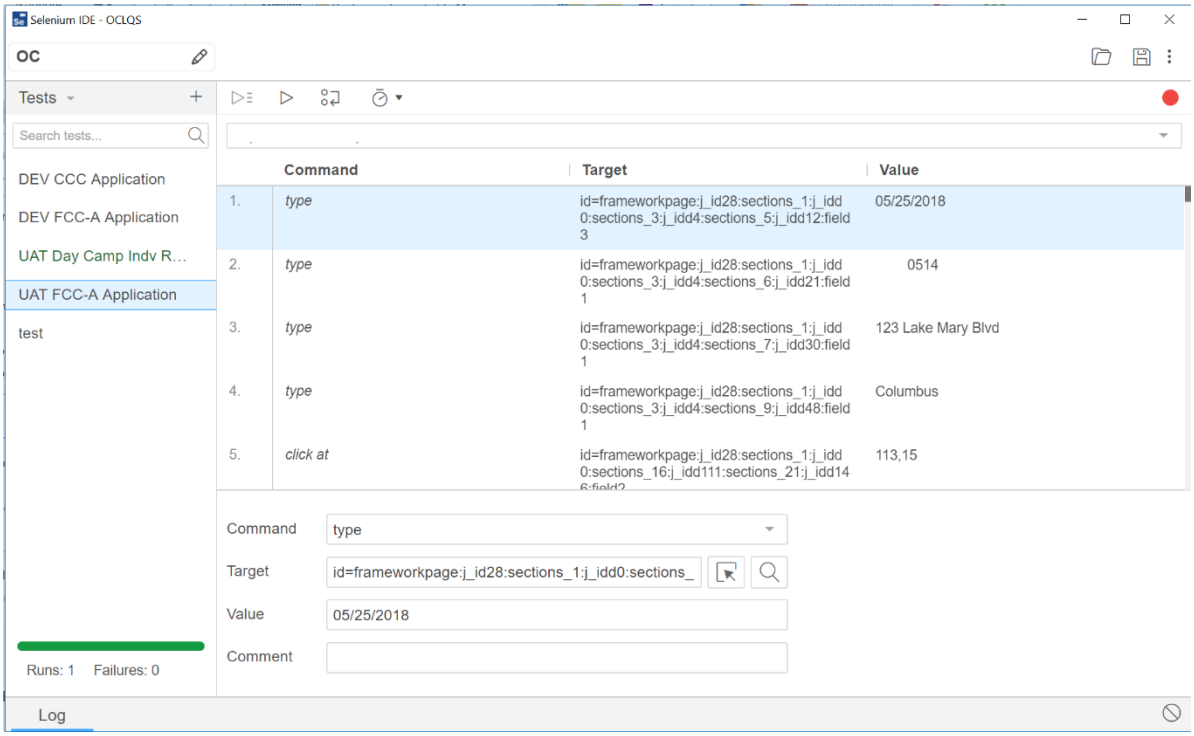

Рисунок 1.5 – Інтерфейс фреймворку Selenium IDE

Katalon являється духовним нащадком фреймворку Selenium. Він перейняв у свого попередника ряд переваг, пов’язаних з інтеграцією процесу тестування. Також даний фреймворк є простим для освоєння і навіть люди, далекі від програмування здатні самостійно запустити новий проект[21].

Приклад графічного інтерфейсу наведено на рис. 1.6.

Проте набір інструментів є доволі обмеженим, вихідний код закритий, мова програмування обмежується Java, а браузери – лише кількома найпопулярнішими. Тому даний фреймворк є зручним, але недостатньо функціональним.

UFT (Unified Functional Testing) – популярний комерційний тестовий фреймворк. Як можна помітити з назви, даний фреймворк позиціонується як універсальний, яким по суті і являється. Даний фреймворк має в собі значний обсяг інструментів для автоматизації тестування API, графічного інтерфейсу, мобільних додатків тощо[21].

Приклад графічного інтерфейсу наведено на рис. 1.7.

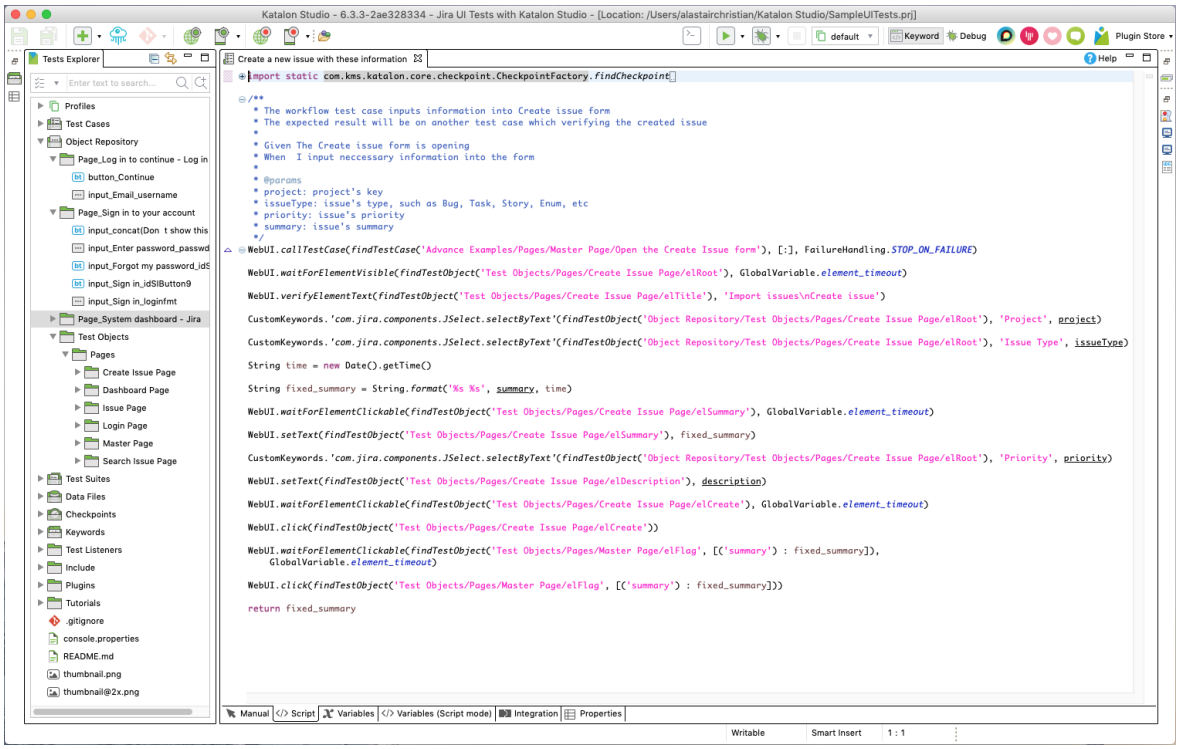

Рисунок 1.6 – Інтерфейс фреймворку Katalon

Проте будучи комерційним проектом з вартістю ліцензії в 800 доларів і працюючи на доволі непопулярній мові VBScript, даний фреймворк пропонує надлишковий функціонал в контексті тестування веб-інтерфейсу за великі гроші.

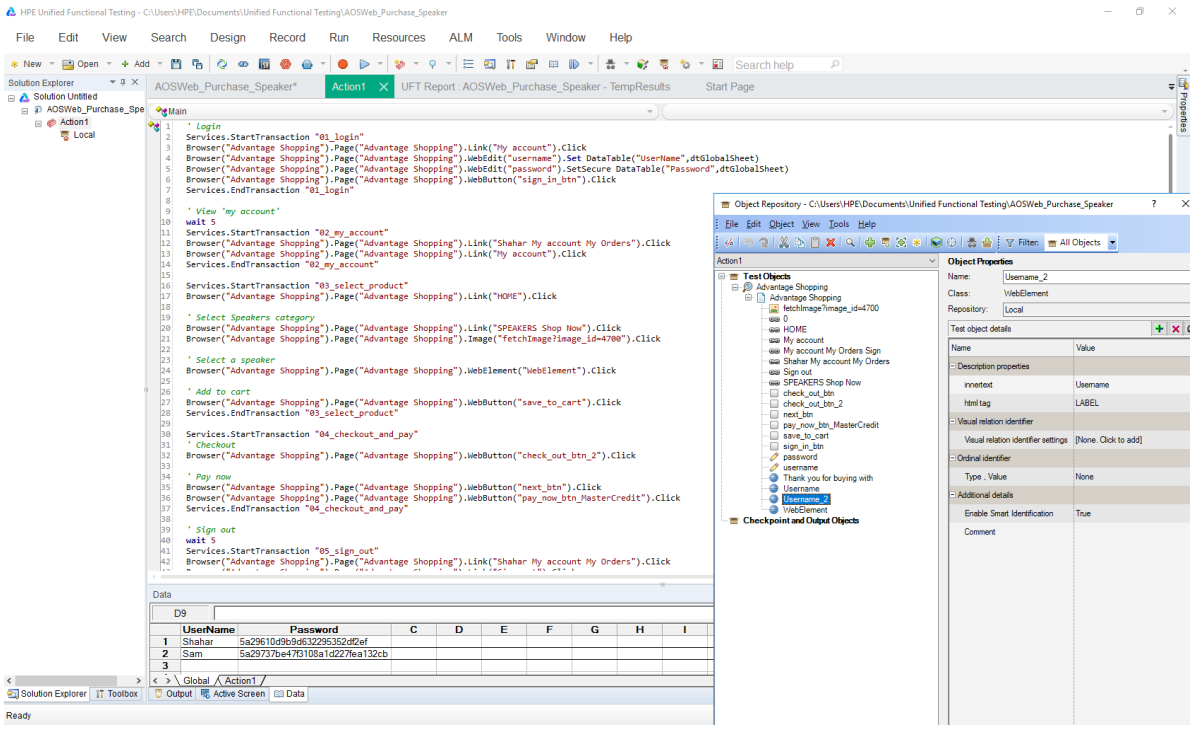

Рисунок 1.7 – Інтерфейс фреймворку UTF

Test Complete являється дуже ефективним інструментом автоматизації, поєднуючи переваги двох попередніх фреймворків. Порівняно з Katalon даний фреймворк має такий самий зручний інтерфейс і низький поріг входження. Порівняно з UTF, він має менший набір функціоналу, але все одно надає можливість тестувати мобільні, десктопні та веб додатки[17]. Лише неймовірно висока ціна та закритий програмний код стає на заваді використанню даного фреймворка як базового.

Приклад графічного інтерфейсу наведено на рис. 1.8.

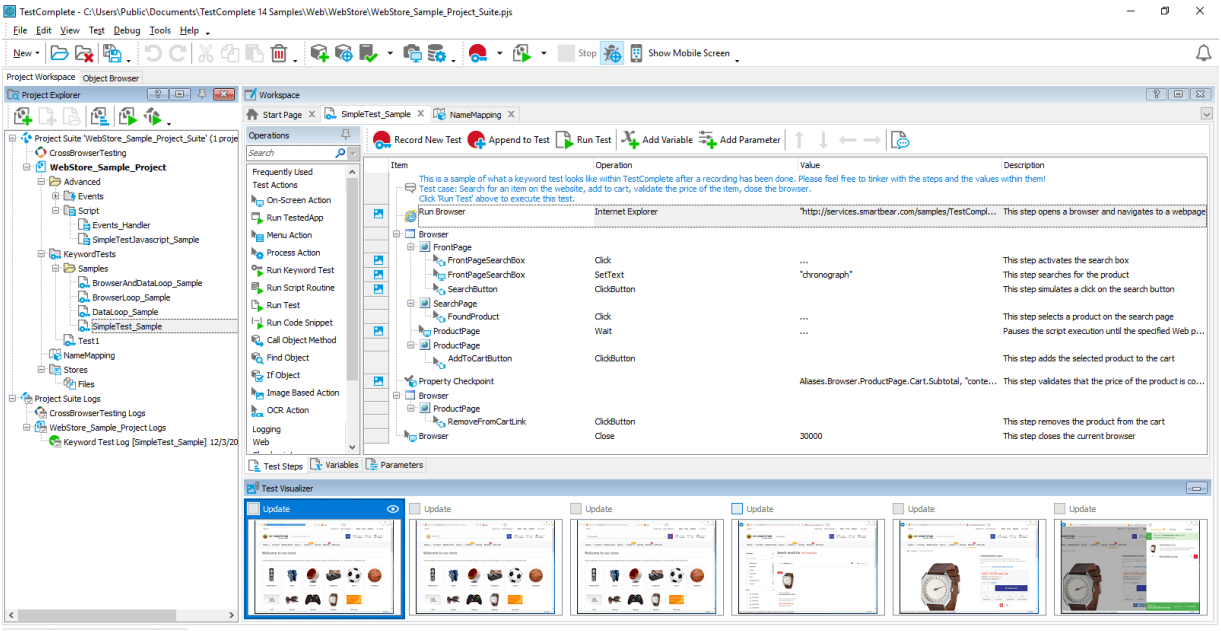

Рисунок 1.8 – Інтерфейс фреймворку Test Complete

Watir фактично можна назвати братом-близнюком для фреймворка Selenium. Схожий набір інструментів, відкритий код, можливість роботи з різними браузерами – все це робить даний фреймворк доволі поширеним, попри такий його недолік як функціонування виключно з мовою Ruby[17].

Приклад графічного інтерфейсу наведено на рис. 1.9.

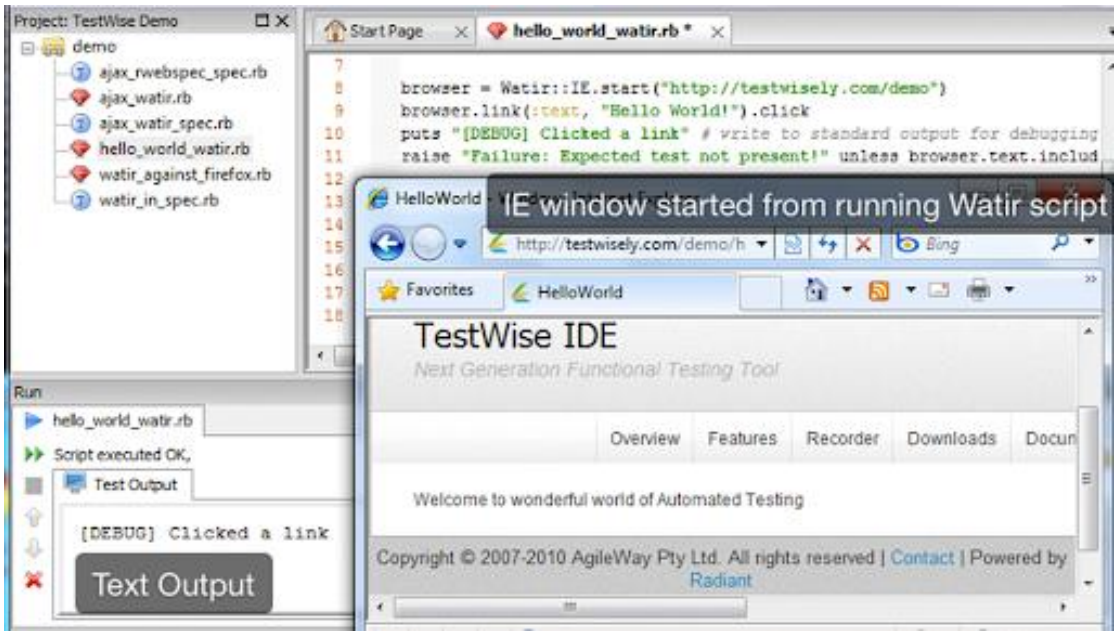

Рисунок 1.9 – Інтерфейс фреймворку Watir

Таким чином, розглянувши основні фреймворки автоматизованого тестування, можна зробити наступні висновки. По-перше, найбільш універсальною мовою використання є Java. По-друге, виходячи з того, що в даному проекті тестування відбувається виключно для веб-сторінки, нема сенсу у впровадженні надлишкових інструментів для тестування інших видів додатків. По-третє, фреймворк повинен мати невисокий поріг входження та давати доступ до широкого набору інструментів.

На основі цього найкращим вибором для базового фреймворка можна вважати Selenide. Будучи нащадком Selenium, даний фреймворк має доступ до всього набору інструментів Selenium.

Проте на відміну від свого пращура, даний фреймворк має ряд переваг, що значно знижують поріг входження користувача. В першу чергу, це реалізація конфігураційних кроків на рівні інструменту. Тобто налаштування браузера, якщо користувачем не вказано зворотнє, виконуються автоматично.

По-друге, засоби фіксації та збереження даних про помилки. У разі, коли тестовий сценарій не проходить, Selenide автоматично генерує і зберігає скриншот, який демонструє екран в момент зупинки тесту.

І остання перевага, хоча по значущості вона може бути і першою, - це нативність команд. Більшість команд адаптовані під сталі розмовні конструкції англійської мови, тому написання тестів для людини, що знає англійську мову стає інтуїтивно зрозумілим. В іншому випадку, Selenide надає вичерпну документацію, яка в простому і зрозумілому вигляді дає інформацію про основні інструменти фреймворку.

2.СПЕЦИФІКАЦІЯ ВИМОГ ДО СИСТЕМИ

2.1 Глосарій

В попередньому розділі було використано декілька термінів, властивих конкретній предметній області, а саме сфері тестування програмного забезпечення. В наступних розділах кількість спеціалізованих термінів буде зростати, тому є необхідність чітко і однозначно описати визначення даних термінів. Для цього необхідно винести інформацію в глосарій.

Глосарій можна поділити на три концептуальні частини: «Основні поняття і категорії предметної області та проекту», «Користувачі системи» та «Вхідні і вихідні документи».

В першій частині глосарію буде наведено список визначень термінів, притаманних предметній області. Варто зазначати, що у вільному доступі є різні версії визначень для одного терміну, проте в даному глосарії буде використано інформацію з офіційного глосарію International Software Testing Qualifications Board.

Окрім загальних понять, перша частина глосарію матиме в собі технічні терміни, використані під час імплементації даного проекту. Пояснення даних термінів спростить подальше розуміння роботи фреймворку.

В пункті «Користувачі системи» буде коротко описано роль та види діяльності кожної відповідної особи, а також зв’язок їх обов’язків зі створюваним фреймворком. Пункт «Вхідні та вихідні документи» в загальних рисах описує призначення та структуру відповідних документів.

Повний глосарій наведено в таблиці 2.1

Таблиця 2.1 – Глосарій

Термін	Опис терміну
1	2
1. Основні поняття і категорії предметної області та проекту	
Тестовий сценарій	Набір вхідних значень, передумов виконання, очікувані результати та постумови виконання програми або тестові умови для перевірки відповідності до певній вимоги.
Тестові артефакти	Набір обов’язкових документів та моделей, які створюються під час проведення тестування і описують процес тестування
Димове тестування	Вибірка із загального числа тестових сценаріїв, що покриває основну функціональність компонента або системи. Проводиться з метою упевнитися, що базові функції програми

	в цілому працюють коректно, без поглиблення в деталі.

Продовження таблиці 2.1

1	2
Регресійне тестування	Тестування вже протестованої програми, що проводиться після модифікації для впевненості в тому, що процес модифікації не вніс або не активовував помилки в областях, що не піддавалися змінам. Проводиться після змін в коді програмного продукту або його оточення.
Метрики тестування	Шкала вимірювань і метод, який використовується для вимірювань ефективності процесу тестування
Відсоток проходження тестів	Відношення кількості тестів, виконання яких було проведено і отримано статус «Пройдено», до загальної кількості проведених тестів
Дефект програмного продукту	Вада в компоненті або системі, яка може привести компонент або систему до неможливості виконати потрібну опцію, наприклад, невірний оператор або визначення даних. Дефект, виявлений під час виконання, може привести до відмов компонента або системи.
Шаблон проектування	Ефективний спосіб вирішення задач проектування програмного забезпечення. Шаблон не є закінченим зразком, який можна безпосередньо транслювати в програмний код. Об'єктно-орієнтований шаблон найчастіше є зразком вирішення проблеми і відображає відношення між класами та об'єктами, без вказівки на те, як буде зрештою реалізоване це відношення.
Тестовий набір	Комплект тестових сценаріїв для досліджуваного компонента або системи, в якому зазвичай постумова одного тесту використовується в якості передумови для подальшого.
Тестова сесія	Один або більше тестовий набір, виконання яких розпочинається користувачем в один момент часу
DOM-дерево	Специфікація прикладного програмного інтерфейсу для роботи зі структурованими документами (як правило, документами XML), що часто використовується в побудові візуального інтерфейсу веб-сторінки
Локатор	Унікальний рядок, що ідентифікує єдиний елемент в структурі DOM-дерева.
Елемент сторінки	Блок, що використовується для компонування сторінки веб-сайту та шаблонів веб-сайту
Анотація	Скорочений запис для виклику методів, реалізованих в сторонній бібліотеці.
Статус тесту	Результат виконання тестового сценарію, зазвичай може мати значення «Пройдено», «Не пройдено». Інколи можливі

	специфічні значення «Не виконано»

Закінчення таблиці 2.1

1	2
IDE	Комплексне програмне рішення для розробки програмного забезпечення. Складається з редактора коду, інструментів для автоматизації складання та відлагодження програм.
Тестові дані	Дані, що отримуються об'єктом тестування з зовнішнього джерела під час проведення тестування. У ролі зовнішнього джерела може виступати апаратне забезпечення, програмне забезпечення або людина.
Ассерт	Частина тестового методу, яка засвідчує відповідність фактичного результату з очікуваним.
Пріоритет тесту	Міра першочерговості, що присвоюється тесту.
Логгер	Частина програмного продукту, що займається відстеженням помилок під час роботи продукту та збереження інформації в разі дефектів його роботи
Крок сценарію	Метод, який інкапсулює в собі функціональність одного кроку тестового сценарію і являється передумовою для старту наступного
2. Користувачі системи	
Тестувальник	Досвідчений фахівець, який приймає участь в тестуванні компонента або системи
3. Вхідні та вихідні документи	
Вимоги до функціоналу	Умови або можливості, необхідні користувачу для вирішення певних завдань або досягнення певних цілей, які повинні бути досягнуті для виконання контракту, стандартів, специфікації, або інших формальних документів
Дистрибутив ПЗ	Версія програмного продукту, що має певний набір функціональності та відповідає вимогам.
Документація для ПЗ	Набір тестових артефактів та програмної документації, на основі якої планується процес тестування і розробляються тестові сценарії.
Звіт про якість ПЗ	Інформація про статус одного або декількох тестових наборів, яка включає в себе інформацію про кількість тестів, відсоток проходження, статус кожного тесту, а також інформацію про помилки у випадку статусу «Не пройдено».
Протестоване ПЗ	Документ, що містить звіт про будь-якому недоліку в компоненті або системі, який може привести компонент або систему до неможливості виконати потрібну опцію.

2.2 Розробка варіантів використання

2.2.1 Діаграма варіантів використання

В першому розділі було детально розглянуто і описано процес тестування програмного забезпечення, автоматизацію якого виконує розроблюваний фреймворк. Тому на даному етапі є необхідність в переході від опису процесу до опису тих варіантів використання, за допомогою яких і буде втілено роботу програмного продукту.

Для цього необхідно побудувати Use-Case діаграму. Use-Case діаграма складається з трьох основних частин: акторів, варіантів використання та зв'язків між ними [9].

Відповідно до керівництва користувача для мови UML, актором може називатись будь-яка сутність, що взаємодіє з системою зовні або множина логічно пов'язаних ролей, що виконуються під час взаємодії з варіантом використання.

Стандартною графічною позначкою на діаграмі являється фігура «людини», під якою записується ім'я суб'єкта. Окрім того суб'єктом може бути не лише людина, а й технічний пристрій, програма або будь-яка інша система, яка може впливати на роботу модельованої системи.

Варіантом використання можна назвати опис множини послідовних дій, які виконуються самою системою для того, щоб актор отримав необхідний результат. Проте ніяким чином не описується те, як буде реалізовано взаємодію суб'єкта з системою.

Стандартною графічною позначкою варіанта використання на діаграмах являється еліпс, всередині якого знаходиться коротка назва варіанта використання або ім'я в формі дієслова з пояснювальними словами.

Зв'язки – третя складова Use-Case діаграми. Вони демонструють відносини між цільовими об'єктами діаграми та дають уявлення про зміст відносин між ними. Зв'язки можуть бути побудовані як між актором та варіантом використання, так і між двома варіантами використання.

В першому випадку єдиним можливим зв'язком є зв'язок асоціації, яка демонструє те, що актор і варіант використання взаємодіють одне з одним. Графічно цей вид зв'язку позначається простою стрілочкою

В другому випадку можливі три варіанти зв'язків: включення, розширення та узагальнення. Зв'язок включення означає, що для базового варіанту використання необхідне виконання і другорядного варіанту. Графічно цей зв'язок демонструється пунктирною лінією з підписом <<include>>.

Зв’язок розширення демонструє що основний варіант використання містить в собі необов’язкові або виключні варіанти, які можна винести в окремий розширюючий варіант використання. Графічно зв’язок розширення відображається пунктирною лінією з підписом <<extend>>.

Зв’язок узагальнення використовується в тому випадку, коли два або більше варіанти використання мають схожі риси в структурі чи поведінці. Таким чином є сенс виділити окремий батьківський варіант використання, поведінку якого будуть наслідувати нащадки. На діаграмі зв’язок узагальнення відображається прямою лінією з пустим трикутником на кінці.

Діаграма варіантів використання для фреймворку автоматизованого тестування продемонстрована на рис. 2.1.

На рис. 2.1 можна помітити, що в даному випадку наявний лише один актор. Це зумовлено тим фактом, що основними користувачами даної системи є члени команди тестувальників.

З урахуванням того, що більшість сучасних команд працюють згідно agile-методологій, які декларують взаємозамінність учасників команди, немає потреби розділяти актора «Lead команди тестувальників» та актора «Тестувальник»[17]. Подібний поділ акторів не допоможе краще розкрити варіанти використання даного фреймворку, а лише призведе до дублювання варіантів використання двома акторами.

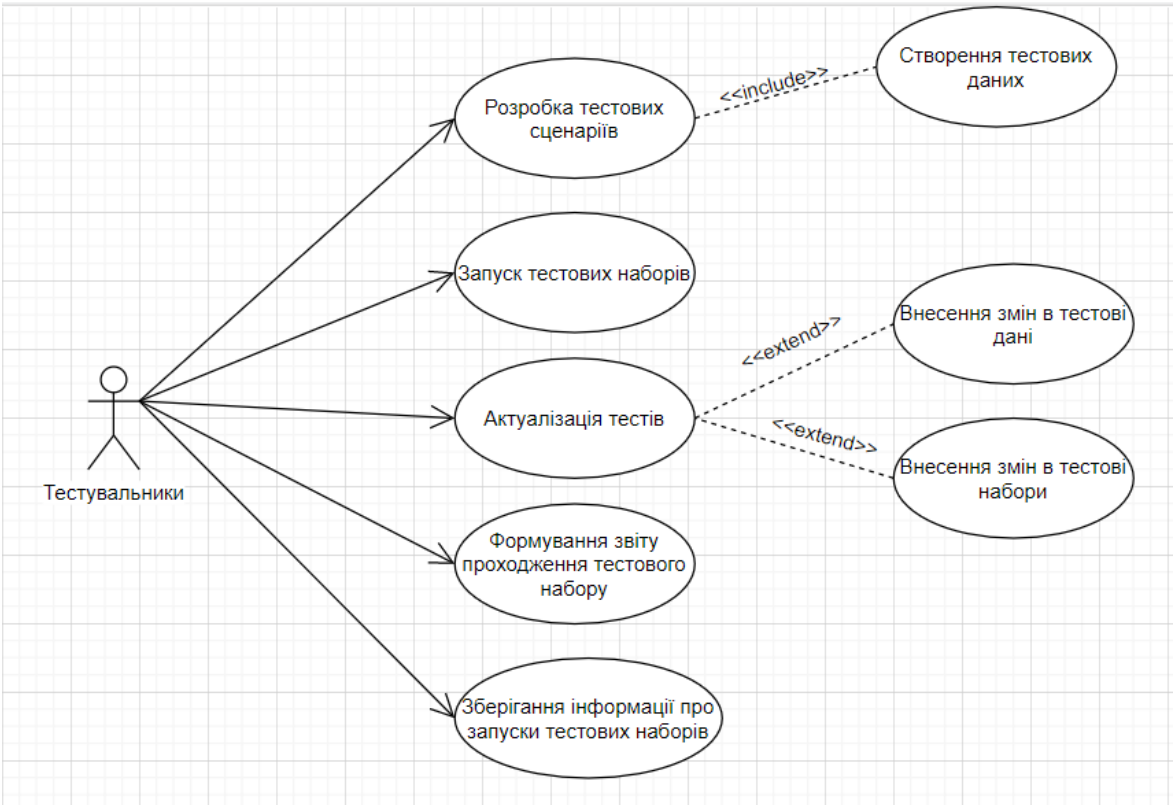

Рисунок 2.1– Діаграма Use-Case фреймворку автоматизації тестування

2.2.2 Специфікація варіантів використання

Першим варіантом використання є «Розробка тестових сценаріїв». Даний варіант декларує можливість актором створення нових тестових сценаріїв для подальшого запуску сценарію в автоматичному режимі. Опис даного варіанту використання розміщено в табл. 2.2.

Таблиця 2.2 – Варіант використання «Розробка тестових сценаріїв»

Контекст використання	Внесення до тестового фреймворку нового тестового сценарію з метою подальшого використання
Дійові особи	Тестувальник
Передумова	Наявність представлень сторінок веб-сайту
Тригер	Поява нової функціональності на веб-сайті
Сценарій	1. Створення кроків тестового сценарію 2. Імплементація кроків тестового сценарію 3. Введення ассертів 4. Додавання тестового сценарію до тестового набору
Постумова	Всі дані, що були в системі і не пов’язані з тестовим сценарієм лишаються незмінними

До попереднього варіанту використання «включним» варіантом використання є «Створення тестових даних». Цей варіант використання свідчить про те, що під час розробки тестових сценаріїв необхідно також сформувати тестові дані, які будуть йти на вхід для кожного з тестів.

Опис даного варіанту використання розміщено в табл. 2.3.

Таблиця 2.3 – Варіант використання «Створення тестових даних»

Контекст використання	Подання тестових даних на вхід тестовому сценарію
Дійові особи	Тестувальник
Передумова	Створено тестовий сценарій
Тригер	Потреба в вводі даних в процесі виконання тестового сценарію
Сценарій	1. Генерація тестових даних 2. Отримання даних в тестовому сценарії
Постумова	Наявний файл з тестовими даними

Другим варіантом використання виділено «Запуск тестових наборів». В цьому варіанті використання описано можливість запускати відповідні тестові сценарії чи тестові набори з певною множиною стартових даних, в результаті чого отримати статус для кожного запущеного тесту.

Опис даного варіанту використання розміщено в табл. 2.4.

Таблиця 2.4 – Варіант використання «Запуск тестових наборів»

Контекст використання	Старт автоматизованого виконання тестових наборів
Дійові особи	Тестувальник
Передумова	Тестові сценарії створено і згруповано в тестові набори
Тригер	Потреба в оцінці програмного продукту відповідно до певних метрик
Сценарій	1. Старт тестового набору
Постумова	Поява інформації про статус тестів в інформаційному полі інтерфейсу

Третім варіантом використання є «Актуалізація тестів». Оскільки в сучасному світі процес розробки програмного забезпечення, в основному, являється ітеративним, то регулярні зміни програмного продукту потребують регулярного апдейту тестового фреймворку. В даному варіанті використання актор має змогу вносити зміни в кроки тестового сценарію та їх імплементацію.

Опис даного варіанту використання розміщено в табл. 2.5.

Таблиця 2.5 – Варіант використання «Актуалізація тестів»

Контекст використання	Внесення змін в тестові сценарії
Дійові особи	Тестувальник
Передумова	Тестові сценарії створено
Тригер	Зміни в існуючому функціоналі програмного продукту, при яких тестові сценарії стають застарілими чи нефункціонуючими
Сценарій	1. Внесення змін в кроки тестового сценарію 2. Внесення змін в механізми кроків сценарію 3. Перевірка ассертів
Постумова	Поява інформації про статус тестів в інформаційному полі інтерфейсу

З попереднім варіантом використання зв’язком розширення пов’язані варіанти використання «Внесення змін в тестові дані» та «Внесення змін в тестові

набори». Оскільки ані тестові дані, ані тестові набори не пов’язані напряму з тестовим сценарієм і актуалізація цих аспектів є необов’язковим в процесі проходження основного варіанту використання, то потрапляють під поняття зв’язку розширення.

Опис варіанту використання «Внесення змін в тестові дані» розміщено в табл. 2.6. Опис варіанту використання «Внесення змін в тестові набори»» розміщено в табл. 2.7.

Таблиця 2.6 – Варіант використання «Внесення змін в тестові дані»

Контекст використання	Внесення змін в тестові дані
Дійові особи	Тестувальник
Передумова	Тестові сценарії створено, тестові дані існують
Тригер	Зміни в існуючому функціоналі програмного продукту, при яких тестові дані втрачають сенс
Сценарій	1. Визначення застарілих даних 2. Заміна необхідних даних
Постумова	Наявний файл з тестовими даними

Таблиця 2.7 – Варіант використання «Внесення змін в тестові набори»

Контекст використання	Внесення змін в тестові набори
Дійові особи	Тестувальник
Передумова	Тестові сценарії створено, тестові набори існують
Тригер	Поява нових тестових сценаріїв, при яких існуючі тестові набори стають надто загальними
Сценарій	1. Переоцінка тестових сценаріїв 2. Групування сценаріїв за тестовими наборами
Постумова	Файли з тестовими наборами існують

Четвертим серед основних варіантів використання є «Формування звіту проходження тестового набору». Даний варіант використання формує детальний звіт і може бути сформований лише для останнього запуск тестового набору. В звіті цього варіанту використання окрім інформації про набір, час запуску та відсотку пройдених тестів є інформація про статус кожного конкретного тесту в наборі, а також в разі помилки є інформація про неї. Опис варіанту використання «Внесення змін в тестові набори» розміщено в табл. 2.8.

Таблиця 2.8 – Варіант використання «Формування звіту проходження тестового набору»

Контекст використання	Формування звіту проходження тестового набору
Дійові особи	Тестувальник
Передумова	Тестовий набір було запущено
Тригер	Тести отримали статус і сформований відсоток проходження.
Сценарій	1. Отримання інформації про останній запуск 2. Генерація звіту в форматі html
Постумова	Звіт з даними про останній запуск тестового набору існує

Останнім варіантом використання є «Збереження інформації про запуск тестових наборів». На відміну від попереднього варіанту використання, даний не формує звіт в зручній для ознайомлення формі. Цей варіант використання свідчить про те, що дані про всі тести, тестові набори та кожен їх запуск зберігається в базі даних. Опис варіанту використання «Збереження інформації про запуск тестових наборів» розміщено в табл. 2.9.

Таблиця 2.9 – Варіант використання «Збереження інформації про запуск тестових наборів»

Контекст використання	Збереження інформації про запуск тестових наборів
Дійові особи	Тестувальник
Передумова	Тестовий набір було створено
Тригер	Тести отримали статус і сформований відсоток проходження.
Сценарій	1. Підключення до бази даних 2. Запуск тестового набору 3. Запис інформації в базу даних
Постумова	Запис про тестову сесію занесена в базу даних

2.3 Специфікація функціональних та нефункціональних вимог

2.3.1 Специфікація функціональних вимог

Як було зазначено раніше, створення діаграми варіантів використання, а також специфікації варіантів використання дає уявлення про основний функціонал, який здатен забезпечити тестовий фреймворк.

На основі цієї інформації можна створити набір функціональних вимог, виконання яких є необхідним для правильної роботи програмного продукту. Для цього варто створити специфікацію функціональних вимог.

Специфікація функціональних вимог наведена в табл. 2.10.

Таблиця 2.10 – Специфікація функціональних вимог

Ідентифікатор вимоги	Назва вимоги (варіанту використання)	Атрибути вимоги	
		Пріоритет	Трудність
1	2	3	4
ФВ-1	Розробка тестових сценаріїв	Обов’язково	Середня
ФВ-2	Створення тестових даних	Рекомендовано	Середня
ФВ-3	Актуалізація тестів	Обов’язково	Середня
ФВ-4	Внесення змін в тестові набори	Обов’язково	Низька
ФВ-5	Формування звіту проходження тестового набору	Обов’язково	Висока
ФВ-6	Внесення змін в тестові дані	Рекомендовано	Низька
ФВ-7	Запуск тестових наборів	Обов’язково	Висока
ФВ-8	Збереження інформації про запуск тестових наборів	Опційно	Середня

В даній специфікації варто надати інформацію про першочерговість впровадження та затрати на імплементацію тих чи інших вимог. Ці дані подано в стовбцях «Пріоритет» та «Трудність» відповідно. Також кожній із вимог присвоєно унікальний ідентифікатор.

Варто зазначити, що формулювання вимог співпадає з назвою відповідного варіанту використання. Деякі з вимог являються більш комплексними, адже базуються на більш комплексних варіантах використання. Не зважаючи на те, що такі вимоги можливо розбити на декілька окремих, з метою збереження їх цілісності, таке розбиття впроваджено не було.

Окрім того, в специфікації функціональних вимог було випущено такий атрибут вимоги, як «Контакт» або «Виконавець». Оскільки в даному проекті існує єдина людина в особі Гаркавого Станіслава Олеговича, яка має необхідну інформацію та реалізує її в проекті, дане поле було випущено з метою запобігання повторів.

2.3.2 Специфікація нефункціональних вимог

Окрім набору виконуваних функцій, будь-який програмний продукт повинен відповідати вимогам експлуатації, стабільності, продуктивності тощо.

Таким чином є необхідність в формуванні окрім специфікації функціональних вимог ще й специфікацію вимог нефункціональних.

Структурно кожна з вимог в даній специфікацій описується схожим чином, як і вимоги функціональні в специфікації функціональних вимог. Кожна нефункціональна вимога отримує унікальний ідентифікатор, описується її пріоритет та трудність впровадження. Аналогічно до специфікації функціональних вимог, в даній специфікації опущено поле «Контакт» або «Виконавець».

Специфікація нефункціональних вимог описана в табл. 2.11.

Таблиця 2.11 – Специфікація нефункціональних вимог

Ідентифікатор вимоги	Назва вимоги (варіанту використання)	Атрибути вимоги	
		Пріоритет	Трудність
1	2	3	4
1. Застосовність			
НВ-1	Час, необхідний для навчання звичайного користувача, не перевищує 24 робочих годин	Обов’язково	Середня
НВ-2	Час, необхідний для навчання досвідченого користувача, не перевищує 40 робочих годин	Рекомендовано	Середня
2. Надійність			
НВ-3	Середній час безвідмовної роботи складає 6 місяців	Обов’язково	Низька
НВ-4	Кількість тестових сценаріїв, які отримали статус «Не пройдено» через дефекти фреймворку мають складати не більше 5% обсягу тестового набору	Рекомендовано	Висока
3. Робочі характеристики			
НВ-5	Виконання одного кроку тестового сценарію не повинно перевищувати 1 секунду	Обов’язково	Середня
НВ-6	Максимальна кількість паралельно запущених тестових наборів на одному комп’ютері дорівнює 5	Рекомендовано	Середня
НВ-7	Максимальна кількість користувачів фреймворку одночасно не може перевищувати	Опційно	Низька

	100		
4. Експлуатаційна придатність			

Продовження таблиці 2.11

1	2	3	4
НВ-8	Код повинен бути написаний відповідно до Java Code Convention	Обов’язково	Середня
НВ-9	Назви елементів сторінки в фреймворку повинні бути недвозначні і співпадати з назвою на веб-сайті	Рекомендовано	Низька
НВ-10	Назви хелперів мають відповідати шаблону: PagenameHelper	Обов’язково	Низька
НВ-11	Назви методів в класах-хелперах повинні бути оформлені за допомогою кемел-кейсу	Обов’язково	Низька
НВ-12	Назви методів в класах-хелперах повинні описувати одним реченням дію з веб-сторінкою, яку реалізовує цей клас	Обов’язково	Низька
5. Проектні обмеження			
НВ-13	Робота зі сторінками веб-сайту повинна бути реалізована відповідно до шаблонів проектування Page Object/Fragment	Обов’язково	Середня
НВ-14	Використана мова програмування Java	Обов’язково	Середня
НВ-15	Версія мови програмування повинна бути Java 8	Рекомендовано	Середня
НВ-16	Тестові сценарії описуються за допомогою фреймворку TestNG	Обов’язково	Низька
НВ-17	Імплементація кроків тестового сценарію відбувається в классах-хелперах	Обов’язково	Висока
НВ-18	Для створення сторінок повинно	Обов’язково	Середня

	бути використано засоби фреймворку Selenide		
НВ-19	Тестові набори повинні бути зібрані в xml-файлах	Обов’язково	Середня
НВ-20	Генерація звіту повинна відбуватись за допомогою репорт-системи Allure	Рекомендовано	Середня

Закінчення таблиці 2.11

1	2	3	4
НВ-21	Фреймворк повинен підтримувати роботу Maven	Обов’язково	Середня
НВ-22	Інформація про тестування повинна заноситись до реляційної бази даних на основі MySQL	Обов’язково	Низька
6. Вимоги до документації, призначеної для користувача, і до системи допомоги.			
НВ-23	Елементи тестових класів повинні використовувати анотації з фреймворку TestNG	Обов’язково	Середня
НВ-24	Елементи класів-сторінок повинні використовувати анотації з фреймворку Selenide	Рекомендовано	Середня
7. Інтерфейси			
НВ-25	Фреймворк повинен підтримувати інтерфейс IDE NetBeans, Intellij Idea та Eclipse	Обов’язково	Середня
НВ-26	При запущеному тестовому наборі, статус поточного проходження повинен відображатись на екрані	Рекомендовано	Середня
НВ-27	Після завершення виконання тестового набору, повинна бути інформація про проходження кожного тесту	Обов’язково	Середня

2.4 Проектування інтерфейсу користувача

Інтерфейс користувача – інтерфейс, що забезпечує комунікацію між користувачем та програмним продуктом. Саме вдало спроектований інтерфейс користувача робить програму зручною у використанні і може виділити її серед безлічі аналогів.

Фреймворки автоматизованого тестування є доволі специфічним видом програмного продукту з точки зору інтерфейсу. Оскільки основними користувачами даних інструментів є тестувальники (і в рідких випадках розробники), то часто такі фреймворки не мають чітко вираженого інтерфейсу користувача.

Найбільш розповсюджена практика інтерфейсу користувача в фреймворках автоматизованого тестування – надбудова для запуску тестових наборів над популярними IDE для тої мови програмування, на якій написано фреймворк. За допомогою IDE відбувається процес створення тестових сценаріїв та наборів, вносяться зміни тощо.

Надбудови над IDE відповідають за відстеження тестових сесій, збереження інформації про проходження, генерацію звіту з тестування тощо.

Вайрфрейм подібної надбудови над IDE Intellij IDEA можна побачити на рис. 2.2.

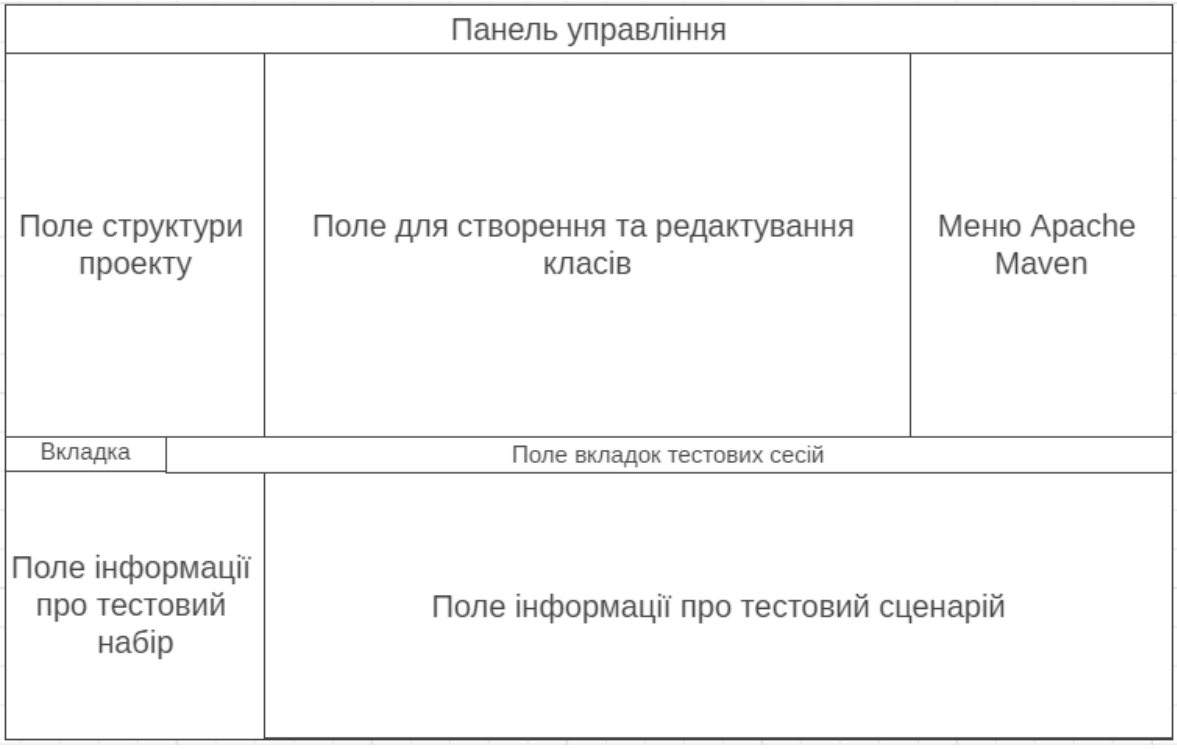

Рисунок 2.2– Макет інтерфейсу IDE Intellij IDEA при застосуванні фреймворку автоматизованого тестування

Інтерфейс розробки залишається повністю ідентичною до класичної IDE.

Біля лівого краю інтерфейсу розробки знаходиться дерево проекту, в якому відображено структуру файлів даного фреймворку. В центрі знаходиться основне

робоче поле, в якому відбувається створення класів, методів, тестів, створюються та модифікуються файли налаштувань, вхідні дані тощо. Біля правого краю вікна знаходиться кнопка для запуску фреймворку автоматизованого тестування за допомогою фреймворку автоматичної збірки Apache Maven.

Фреймворк Apache Maven є невід'ємною частиною для якісної і стабільної роботи тестових фреймворків. Apache Maven побудований на основі принципу опису структури проекту за допомогою мови POM, яка в свою чергу є похідною від мови XML.

Використання даного фреймворку полегшує роботу з запуском тестових наборів та генерації звіту по запуску тестових набору. Оскільки в його структурі закладено чітко визначений цикл життя, то використання меню Apache Maven для роботи фреймворку автоматизованого тестування є життєво необхідним.

Інтерфейс виводу інформації про запуск на відміну від інтерфейсу розробки є унікальним і відрізняється від стандартної консолі виводу IDE. Він складається із однієї або більшої кількості вікон тестових сесій. В кожному такому вікні відображається інформація про проходження тестових наборів, а також інформація про статус окремого тестового сценарію.

Біля лівого краю даного інтерфейсу знаходиться набір кнопок для контролю проходження тестових сценаріїв. За їх допомогою можна зупинити і перезапустити тестові набори, внести додаткові команди під час виконання конкретного тесту.

Правіше від цих кнопок знаходиться вікно статусів. В ньому відображається статус тестового набору або сценарію, який було виконано, а також відмічаються сценарії, виконання яких ще не почалось.

З правого боку розташоване велике інформаційне вікно. При виборі окремого тестового сценарію, в даному вікні відображається повна інформація про перебіг виконання тесту, а також інформація про помилки в разі статусу сценарію «Не пройдено».

3. ПРОЕКТНІ ТА ТЕХНІЧНІ РІШЕННЯ

3.1 Логічна постановка

Процес тестування програмного забезпечення є невід'ємною складовою процесу розробки. Протягом розробки кількість імплементованого функціоналу неухильно зростає, який в свою чергу вимагає покриття тестами. Окрім того, кожна ітерація процесу розробки вимагає додаткового регресійного тестування. Таким чином затрати людських ресурсів, задіяних на перевірку старих тестів і створення нових, зростають з кожним пройденим спринтом[20].

Якщо мова йде про розробку веб-сайту, то процес тестування стає складнішим ніж у випадку з настільним чи мобільним додатком. Причиною тому є той факт, що практично паралельно відбувається два процеси тестування: тестування для back-end і тестування для front-end. І для кожного з цих процесів є свій набір функціональних та нефункціональних вимог, які повинні бути покриті тестами та перевірені[34].

Окрім того в процесі росту проекту кількість рутинних і одноманітних тестів невпинно збільшується. Більшість з них виконуються виключно на етапі регресії і їх результат в основному незмінний. Це призводить до нераціонального розподілу трудових ресурсів і як наслідок погіршення продуктивності працівників.

Найкращим варіантом вирішення даних проблем є фреймворк автоматизованого тестування. Даний варіант не є ідеальний, адже на етапі розробки і впровадження він вимагає значних затрат ресурсів. Фактично, він є своєрідним «проектом всередині проекту». Тому його імплементація є доцільною лише у випадку значної кількості рутинних тестів, проходження яких відбувається регулярно і вимагає значних затрат часу[13].

Значна частина тестів, які перевіряють працездатність інтерфейсу користувача веб-сайту, задовольняє цим умовам. В більшості випадків тестування UI полягає в взаємодії з елементами веб-сторінки та вводу необхідних даних. Інакше кажучи, цей процес складається з повторення дій над елементами сторінки в різних послідовностях. Це є першим доказом необхідності створення тестового фреймворку.

Окрім випадків, коли сайт являє собою візитівку, сайт є набором взаємопов'язаних сторінок, кожна з яких виконує набір власних функцій і може бути проміжною ланкою в процесі виконання функцій інших сторінок. Таким чином навіть простий сайт ломбарду може потребувати до півсотні тестів, які покрили б основний функціонал.

Таким чином, маємо значну кількість тестів, які часто виконуються за допомогою повторення подібного набору дій. Це свідчить про те, що для тестування front-end складової веб-сайту створення фреймворку автоматизованого тестування є доцільним.

Сам фреймворк повинен відтворити кроки, які мав би зробити тестувальник в процесі виконання тестового сценарію. У випадку ручного виконання можна було б назвати три складові даного процесу: актор (тестувальник), дії які актор робить (кроки тестового сценарію) та інтерфейс, в якому відбуваються дії актора (веб-сайт, який тестується) [25]. Всі ці три частини також мають бути представлені в складі фреймворку для автоматизованого тестування. Для цього використаємо принципи моделювання.

В першу чергу, необхідно створити модель інтерфейсу. Формально, сучасні засоби для роботи з веб-сайтами дозволяють розробити тестові сценарії таким чином, що моделювання сторінки не є обов'язковим. За допомогою локаторів можна звернутись до елементу в DOM-дереві прямо в процесі виконання тестового сценарію. Проте такий підхід є нераціональним, адже це зменшує можливості до повторного використання коду і збільшує обсяг кожного окремого тестового сценарію.

Найбільш зручним і загальновизнаним підходом для моделювання веб-сторінки є використання шаблону проектування Page Object/Fragment.

Загальна концепція всіх шаблонів на основі Page Object полягає в розділенні логіки тестового сценарію (що треба перевірити) та її реалізації (якими чином це перевіряється). Наприклад, маємо тестовий сценарій «Користувач вводить неправильний логін чи пароль, натискає кнопку входу і бачить повідомлення про помилку». Тут описана логіка тесту, а сама його реалізація вже має в собі такі дії як пошук необхідних елементів, виведення повідомлення про помилку тощо. Тому у випадку, якщо зміниться, наприклад, локатор поля вводу даних, то помилка ніяк не вплине на логіку тесту. Таким чином тести стають більш гнучкими[30].

Реалізація даного шаблону зазвичай полягає в створенні класів відповідних сторінок. В такому класі зазвичай зберігаються локатори елементів сторінки, також можливе зберігання методів для роботи з цими класами. Проте з метою збереження атомарності класів, рекомендується винести методи для роботи з кожним класом в окремий клас.

Page Fragment є підвидом класу Page Object, проте замість сторінки в ньому моделюється лише її певний фрагмент, який зазвичай присутній на декількох

різних сторінках. Таким чином запобігається дублювання коду і покращуються можливості повторного використання.

Другим етапом, необхідним для повного моделювання системи є актор. У випадку ручного тестування актором являється сам тестувальник. У випадку автоматизованого фреймворка його заміняють відповідні класи-хелпери.

Хелпери – класи, використання яких почалось з розвитком шаблону Page Object. Для кожної сторінки створюється окремий хелпер, в якому зберігаються всі методи для роботи з цією сторінкою. Кожен із цих методів реалізує частинку функціоналу, яку можна провадити на цій сторінці.

І останнім аспектом, необхідним для створення повноцінного фреймворку автоматизованого тестування є ті кроки, які повинен відтворити актор для проходження тестового сценарію.

В тестовому сценарії створюється єдиний об’єкт класу хелпер (зазвичай хелпер сторінки, з якої починається сценарій). В процесі тестування в цьому хелпері викликаються методи, які відповідають певним крокам тестового сценарію. Таким чином будується ланцюжок станів від початкового стану першого хелпера (зазвичай це стан, коли сторінка відкрита) до кінцевого стану останнього хелпера, коли всі інструкції були виконані. Після цього за допомогою ассерту відбувається перевірка відповідності очікуваного та отриманого результату.

Таким чином використання саме такого поділу є найкращим рішенням. Окрім вищезазначених переваг на момент розробки, подібний підхід дає довгострокові дивіденди. Зокрема, реалізація класів сторінок та відповідних класів-хелперів на ранньому етапі дозволить значно прискорити створення тестових сценаріїв на пізніх етапах А розподіл функціональності дозволяє реагувати на проблеми роботи фреймворка з меншими затратами ресурсів, оскільки в разі дефекту нема потреби вносити зміни в усі складові фреймворка.

Таким чином, використання подібної архітектури фактично нівелює основні недоліки впровадження автоматизованого тестування, а саме значні обсяги часу, необхідного для впровадження сценаріїв, і людських ресурсів для підтримки працездатності фреймворка.

3.2 Проектування структури бази даних

3.2.1 Концептуальне інфологічне проектування

Процес тестування являє собою постійний процес, який нечасто звертає увагу на результати отримані значний час тому. В більшості випадків важливу роль відіграють результати тестування протягом останніх кількох днів чи тижнів.

Це зумовлюється тим, що програмний продукт, який розробляється, постійно оновлюється і нові версії з’являються регулярно. Тому дані про тестування відносно швидко стають неактуальними.

Така ситуація є стандартною для робочого процесу, але часом ускладнює формування звіту. Наприклад, команди які працюють згідно методології Kanban можуть мати значний період часу між демонстраціями звіту за два спринти, адже суть цього підходу саме в спринтах, протяжність яких визначається складністю задач. В таких випадках для команди тестувальників демонстрація виконаної роботи за спринт може ускладнитись, адже інформації про відсоток проходження тестів кількамісячної давнини вже може і не існувати.

Автоматизація тестування глобально не позбавлена цієї проблеми. Більшість фреймворків, які доступні на ринку, не зберігають інформацію про запуски окрім останнього.

Частково вирішення цієї проблеми знайдено в системах, що забезпечують безперервну інтеграцію. Яскравими прикладами таких систем можна назвати Jenkins, TeamCity тощо.

Дані системи надають можливість запуску автоматизованих тестів через систему Apache Maven. Процес тестування відбувається на боці сервісу, але в загальних рисах не відрізняється від процесу тестування на локальному комп’ютері.

Перед запуском тестів в системах безперервної інтеграції необхідно виставити ряд параметрів проекту, який має тестуватись, зокрема його версію та набір доступний набір функціоналу, а також необхідно обрати тестовий набір, який буде виконуватись[9].

Як і в випадку з локальним запуском, після завершення тестування буде сформовано звіт за допомогою системи Allure Report. Особливістю систем безперервної інтеграції в контексті тестування є той факт, що після виконання тестів в такій системі звіт про тестування буде збережено в розділі попередніх запусків[31]. Таким чином можна зберігати інформацію про двадцять-тридцять попередніх запусків.

В переважній більшості проектів використання систем безперервної інтеграції є достатнім рішенням даної проблеми. Проте неможливість її використання без доступу до вихідного коду самого програмного продукту і доволі невеликий обсяг збережених даних в певних випадках може знівелювати всі переваги використання цих систем.

В таких ситуаціях доволі нетрадиційним, але не менш надійним способом збереження інформації може стати база даних.

Основною метою цієї бази даних є збереження основної інформації про тестові сценарії, тестові набори та їх запуск, зокрема час запуску та відсоток проходження тестів, протягом значного проміжку часу.

Таким чином можна виділити три основні групи даних, які будуть описані в базі даних: інформація про тестові сценарії, набори та сесії. Список всіх елементів зручно буде подати у вигляді словника даних, в якому буде зазначено ідентифікатори, тип та призначення кожного елемента. Словник даних наведено в табл. 3.1.

Таблиця 3.1 – Словник даних

№	Найменування елемента	Ідентифікатор	Тип і довжина	Призначення елемента
1	2	3	4	5
1.	Ім’я тестового сценарію	Test_Name	Строка, до 50 символів	Збереження інформації про назву тестового сценарії
2.	Назва статусу	Status_Name	Строка, до 20 символів	Збереження інформації про статус тестового сценарію
3.	Назва тестового набору	Test_Suit_Name	Строка, до 30 символів	Зберігання інформації про назву тестового набору
4.	Опис тестового набору	Test_Suite_Description	Строка, до 100 символів	Зберігання опису тестового набору
5.	Відсоток проходження	Pass_Rate	Дробове число	Зберігання інформації про відсоток пройдених тестів в тестовому наборі
6.	Час запуску	Run_Time	Дата і час	Зберігання інформації про час і дату запуску тестової сесії

Виходячи з даних наведених в табл. 3.1, а також мети для якої розробляється база даних, можна зробити висновок, що проектована база матиме три основні сутності для зберігання даних про тестові сценарії, тестові набори і тестові сесії.

Окрім трьох основних сутностей, є сенс винести окремо сутність для зберігання даних про статус тестових сценаріїв. Дана сутність буде заповнена одноразово і лишатиметься незмінною, виконуючи роль словника даних.

Тепер необхідно розглянути кожну сутність окремо.

Сутність «Status» являється словником даних, який несе в собі інформацію про можливі статуси тестового сценарію. Тому окрім первинного ключа Status_ID дана сутність має лише атрибут Status_Name.

Сутність «Test» зберігає в собі необхідну інформацію про кожен з тестів, який автоматизовано за допомогою тестового фреймворка. Дана сутність являється значно спрощеною версією тестового сценарію, згідно якого і розробляється тест. Окрім первинного ключа Test_ID, тут наявні атрибути Test_Name і зовнішні ключі Status_ID та Test_Suit_ID.

Сутність «Test Suit» зберігає інформацію про тестові набори. Окрім первинного ключа Test_Suit_ID, дана сутність має атрибути Test_Suit_Name і Test_Suit_Description.

Атрибути цієї сутності потребують додаткового пояснення. В першу чергу сутність «Test Suit» на відміну від сутності «Test» має атрибути для опису. Це зроблено через те, що згідно Java Code Convention назви тестових методів будуються згідно такого шаблону «назваФункціоналу_ОчікуваніВхідніДані_ОчікуванийРезультат», який фактично забезпечує розуміння тестового сценарію. Натомість тестові набори не мають таких вимог і потребують додаткової інформації.

Сутність «Test Session» дає інформацію про тестову сесію, інакше кажучи зберігає дані про те, коли і який тестовий набір було запущено і який відсоток проходження тестів у даного набору. Окрім первинного ключа Test_Suit_ID, сутність має зовнішній ключ Test_Suit_ID і атрибути Pass_Rate і Run_Time.

Оскільки розроблювана база даних не взаємодіє з бізнес-інформацією і переважна більшість її атрибутів є назвами та описами тих чи інших тестових артефактів, то рівень обмежень, які можуть бути до неї застосовані обмежується рівнем атрибута.

Обмеження атрибутів сутностей наведено в табл. 3.2.

Таблиця 3.2 – Обмеження атрибутів сутностей

№	Ім’я атрибуту	Допустимі значення	Формат	Умова	Значення за замовчуванням
1	2	3	4	5	6
1.	Test_ID	Ціле, більше нуля, не NULL	9(10)	-	1, або на 1 більше від попереднього значення

Закінчення таблиці 3.2

1	2	3	4	5	6
2.	Status_ID	Ціле, більше нуля, не NULL	9(10)	-	1, або на 1 більше від попереднього значення
3.	Test_Suit_ID	Ціле, більше нуля, не NULL	9(10)	-	1, або на 1 більше від попереднього значення
4.	Test_Session_ID	Ціле, більше нуля, не NULL	9(10)	-	1, або на 1 більше від попереднього значення
5.	Test_Name	Не NULL, до 50 символів	X(50)	Значення унікальні	
6.	Status_Name	Не NULL, до 20 символів	X(20)	Значення унікальні	
7.	Test_Suite_Name	Не NULL, до 30 символів	X(30)	Значення унікальні	
8.	Test_Suite_Description	Не NULL, до 100 символів	X(100)	Значення унікальні	
9.	Run_Time	Не NULL, формат часу і дати	9(4).9(2).9(2) 9(2):9(2):9(2)	-	
10.	Pass_Rate	Дробове число	9(2)	Значення в рамках від 0 до 100	

3.2.2 Проектування логічної моделі даних

Після визначення елементів бази даних і групування їх в словник даних, попередньої розробки складу сутностей та оформлення обмежень, настає час розробки логічної моделі бази даних.

Основним етапом розробки логічної моделі є утворення зв’язків між сутностям і подальше приведення їх до третьої нормальної форми бази даних. Основним видом зв’язку, використаним в базі даних є зв’язок 1:n, або один до багатьох. Цьому зв’язку відповідає ситуація, коли одному екземпляру сутності А може відповідати багато екземплярів сутності В.

Зв’язок один до багатьох поєднує наступні сутності: «Status» і «Test», «Test» і «Test Suit», «Test Session» і «Test Suit». В першому випадку кожен статус може

бути присвоєно багатьом тестам, але кожному тесту може бути присвоєно лише один статус. В другому випадку один тестовий набір може включати безліч тестів, але кожен тест має бути включеним лише до одного набору. В третьому ж випадку одна тестова сесія включає в себе запуск одного тестового набору, проте кожен тестовий набір може брати участь в багатьох тестових сесія.

Таким чином після формування сутностей та зв’язків між ними можна побудувати схему бази даних в форматі IDEF1X. Схема бази даних в форматі IDEF1X наведена на рис. 3.1.

Після створення попередньої схеми бази даних необхідно пересвідчитись в тому, що витримуються норми перших трьох нормальних форм для створеної бази даних. Виконання цих норм буде підтвердженням того, що сутності і зв’язки між ними сформовано коректно, а отже і сама база даних буде ефективно виконувати свої функції.

Перша нормальна форма свідчить про те, що таблиця не повинна мати стовбців, що повторюються, або стовбців, які не є атомарними (мають більш ніж одне значення). Для спроектованої бази даних ці вимоги виконуються і частково прописані в обмеженнях атрибутів.

Друга нормальна форма полягає в тому, що в таблиці кожен стовбець повинен залежати від первинного ключа. В базі даних можна помітити, що кожна таблиця має нескладений первинний ключ, що автоматично свідчить про повну функціональну залежність кожного атрибуту таблиці від її первинного ключа.

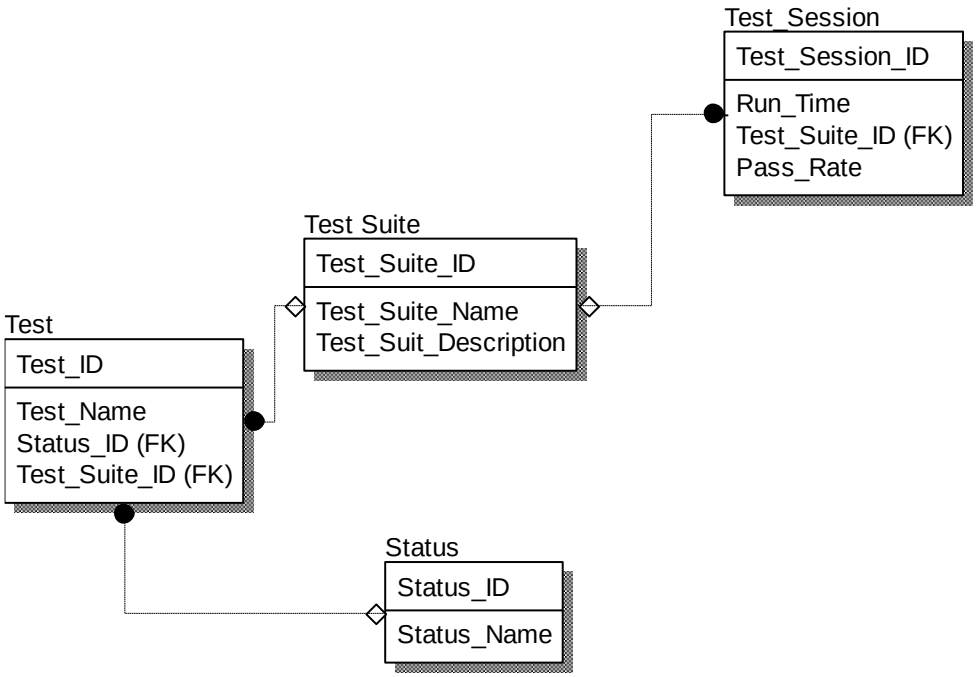

Рисунок 3.1– Схема IDEF1X логічної моделі бази даних

Третя нормальна форма полягає в тому, що в таблиці кожен стовбець повинен залежати ТІЛЬКИ від первинного ключа. Інакше кажучи, в таблицях повинна бути відсутня транзитивна залежність між атрибутами, яка спричиняє надлишковість таблиці. Винесення даних про статус тестових сценаріїв в окрему таблицю привів до того, що база даних відповідає умовам третьої нормальної форми.

3.2.3 Проектування фізичної моделі бази даних

Фізична модель бази даних визначає спосіб розміщення даних в середовищі зберігання і засоби доступу до цих даних, які підтримуються на фізичному рівні. Зазвичай побудова фізичної моделі відбувається на основі логічної моделі з внесенням деяких змін в зв’язки між сутностями.

В першу чергу під час переходу від логічної моделі бази даних до фізичної не можуть використовуватись зв’язки багато до багатьох, а також категоріальні зв’язки.

Категоріальні зв’язки з’являються у тих випадках, коли сутність визначає цілу категорію об’єктів. В такому випадку створюється сутність, яка описує саму категорію, і сутності-нащадки, які описують елементи цієї категорії. Між такими сутностями і встановлюється категоріальний зв’язок.

Розглянувши рис. 3.1 і пункт 3.2.2 можна помітити, що всі наявні в логічній моделі бази даних зв’язки являються зв’язками типу один до багатьох, а отже при створенні фізичної моделі бази даних зв’язки порушені не будуть.

Іншою характерною рисою для фізичної моделі бази даних є відображення таких об’єктів бази даних, як подання, індекси, тригери хранимі процедури тощо. Проте база, що проектується, має просту структуру і використання вищеназваних об’єктів буде надлишковим, тому вони також не вносять зміни в фізичну модель.

Важливим аспектом в процесі створення фізичної моделі є уточнення типів даних і правильне застосування обмежень атрибутів в стовбцях відповідних таблиць фізичної моделі. Звернувши увагу на всі вищезазначені моменти було спроектовано остаточну версію фізичної моделі бази даних.

Фізична модель бази даних зображена на рис. 3.2.

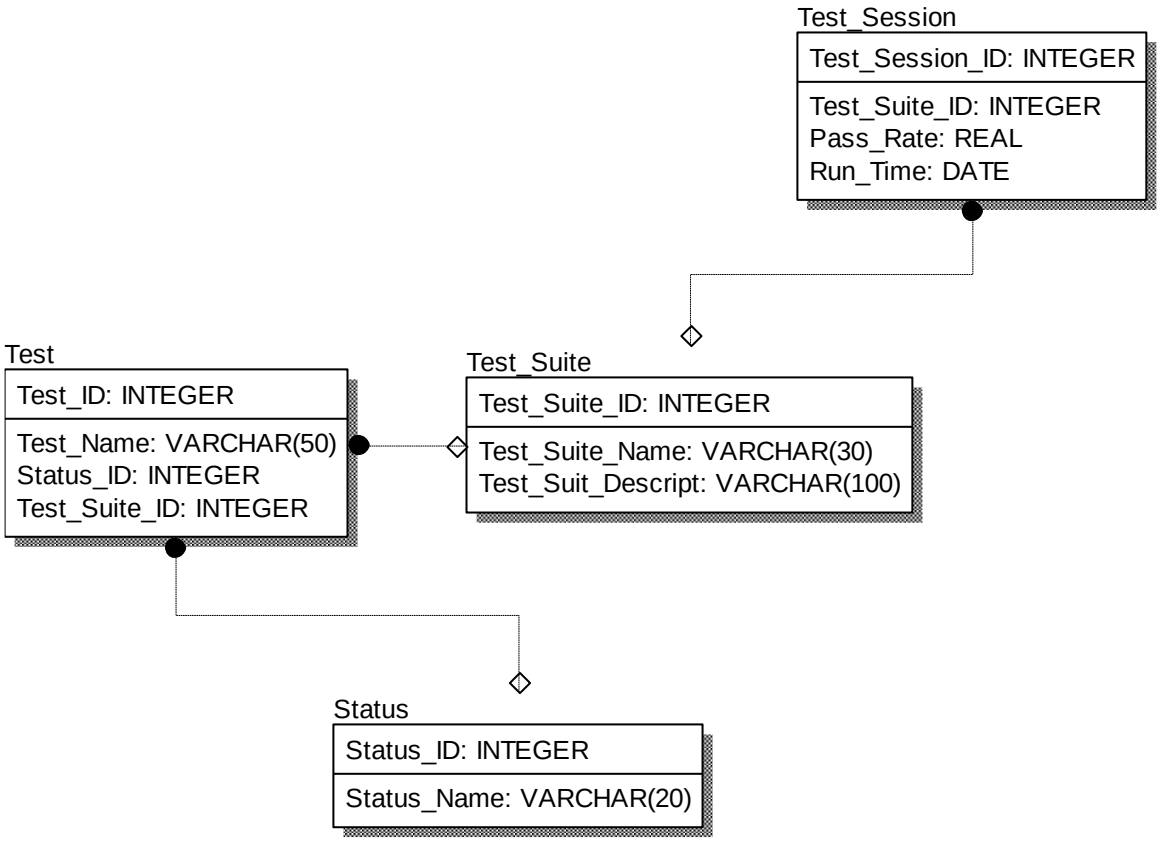

Рисунок 3.2– Схема IDEF1X фізичної моделі бази даних

3.3 Проектування програмного забезпечення

Програмне забезпечення зазвичай являє собою великий набір різноманітних компонентів, які так чи інакше взаємодіють між собою. Для уявлення структури моделі програмного забезпечення призначена діаграма класів. Ця діаграма наглядно відображає існуючі класи та зв’язки між ними. Розглянемо діаграму класів розроблюваного фреймворку автоматизованого тестування.

Діаграма класів наведена на рис. 3.3.

В першу чергу необхідно відзначити, що діаграма на рис. 3.3 є схематичною, адже в ній відсутня інформація про атрибути кожного з класів та його методи. Це було зроблено з метою забезпечення зручної презентації діаграми. Таким чином на ній представлені лише класи та зв’язки між ними. Інформація про методи та атрибути кожного класу буде описана нижче.

В першу чергу на рис. 3.3 можна помітити два базові класи, які є абстрактними: BaseHelper, BaseUIPage. Від кожного з цих класів наслідуються відповідно класи сторінок та хелперів . Розглянемо кожен з базових класів окремо.

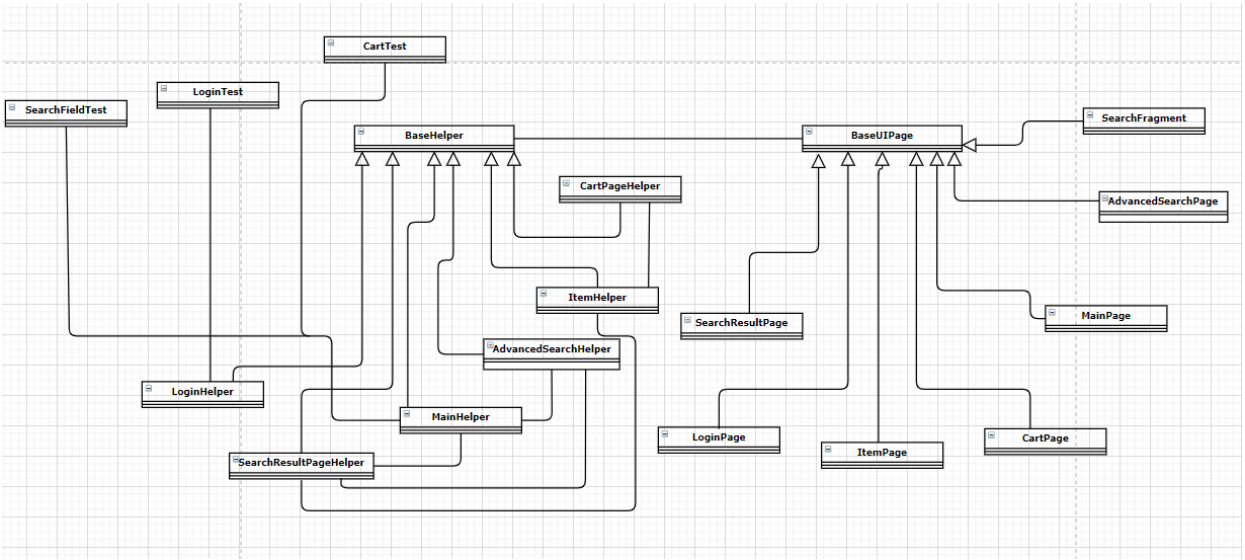

Рисунок 3.3 – Діаграма класів

Клас BaseHelper – базовий клас для всіх класів хелперів. В ньому реалізовано єдиний атрибут Page з модифікатором доступу private, який є нащадком класу BaseUIPage і являється тою сторінкою, на якій працює клас-хелпер.

Даний клас містить два методи refresh() і waitPageLoading(long time, TimeUnit timeUnit). Перший слугує для перезавантаження сторінки, другий являється методом очікування, який слугує для очікування завантаження сторінки і таким чином стабілізує роботу тестового фреймворку.

Клас BaseUIPage – базовий клас для класів сторінок. В ньому існує один атрибут loadableElement типу SelenideElement з модифікатором final. Даний атрибут використовується в конструкторі – єдиному методі даного класу, в якому відбувається завантаження сторінки за цим атрибутом.

Саме від цих двох абстрактних класів наслідуються більшість класів тестового фреймворку. Розглянемо їх детальніше.

Першим великим набором класів є класи сторінки. Кожен з таких класів є представленням справжньої веб-сторінки у вигляді java-класу. Основною особливістю цієї групи класів є відсутність в них методів окрім конструктора, який звертається до конструктора класу BaseUIPage і слугує для завантаження контенту відповідної сторінки. Для цього також в кожному із класів представлено константне поле FIRST_ENTRY.

Список полів усіх класів сторінок фреймворку представлено в табл. 3.3.

Таблиця 3.3 – Поля класів сторінок

Назва елементу	Тип даних	Опис
1	2	3
1. Клас CartPage		
listOfGoodsInCart	ElementsCollection	Список товарів в корзині
quantityField	SelenideElement	Поле, що відображає кількість одиниць товару
numberOfGoods	ElementsCollection	Випадаючий список для вибору кількості товару
removeItemButton	SelenideElement	Кнопка видалення товару
removeMessage	SelenideElement	Повідомлення про те, що товар було видалено
2. Клас ItemPage		
addToCartButton	ElementsCollection	Кнопка «Додати в корзину»
3. Клас LoginPage		
signIn	SelenideElement	Кнопка входу
emailInputField	SelenideElement	Поле для введення електронної пошти
passwordInputField	SelenideElement	Поле для введення паролю
signInButton	SelenideElement	Кнопка «Увійти»
wrongDataMessage	SelenideElement	Повідомлення про те, що було введено некоректну інформацію
profileButton	SelenideElement	Кнопка профілю користувача
continueButton	SelenideElement	Кнопка продовження входу користувача
4. Клас MainPage		
advanceSearchButton	SelenideElement	Кнопка розширеного пошуку
searchFragment	SearchFragment	Фрагмент пошуку
5. Клас AdvancedSearchPage		
inputField	SelenideElement	Поле вводу для пошуку
buyItNowButton	SelenideElement	Кнопка купівлі товару
searchButton	SelenideElement	Кнопка пошуку
6. Клас SearchResultPage		
searchField	SelenideElement	Поле пошуку в звичайному режимі
searchButton	SelenideElement	Кнопка пошуку
searchedGoodsName	ElementsCollection	Колекція, знайдених товарів за допомогою звичайного пошуку
advanceSearchedGoods	ElementsCollection	Колекція, знайдених товарів за допомогою розширеного пошуку

Класи хелпери являються протилежністю класам сторінок. Коли другі не мають методів, окрім конструкторів, то перші не мають полів, окрім поля isPassed типу Boolean, яке використовується для збору інформації про виконання ассертів та подальше використання для отримання відсотку проходження тестів.

Всі методи класів хелперів мають схожість в сигнатурах. В першу чергу всі методи мають тип значення, що повертається, класу цього хелперу. Окрім цього на вхід не подається даних, а на виході повертається цей самий об’єкт класу хелпера або об’єкт іншого хелперу у випадку, коли після виконання методу відбувається перехід на іншу сторінку. Всі методи мають модифікатор доступу public.

Єдиним виключенням є методи-ассерти. Оскільки основною функцією даних методів є підтвердження або спростування успішного проходження тестового сценарію та занесення отриманої інформації в базу даних, то на вхід подається необхідна конфігураційна інформація для встановлення зв’язку та передачі необхідних даних до бази.

Список методів усіх класів хелперів наведено в табл. 3.4.

Таблиця 3.4 – Поля класів хелперів

Назва методу	Значення, що повертається	Опис
1	2	3
1. Клас CartPageHelper		
assertTheNumberOfGoodsInCart	CartPageHelper	Ассерт, що засвідчує вказану кількість товарів в корзині
removeGoodFromCart	CartPageHelper	Метод, що видаляє товар з корзини
assertThatGoodWasRemoved	CartPageHelper	Ассерт, що підтверджує видалення товару з корзини
2. Клас ItemHelper		
addToCartButtonClick	CartPageHelper	Метод, що натискає на кнопку «Додати в корзину» і переводить на сторінку корзини
3. Клас LoginHelper		
open	LoginHelper	Метод відкриття сторінки логіну
login	LoginHelper	Метод для входу

loginWithWrongPassword	LoginHelper	Метод для входу з хибними даними

Закінчення таблиці 3.3

1	2	3
assertTharWrongDataMessageAppears	LoginHelper	Ассерт, що підтверджує появу повідомлення про спробу входу з невалідним логіном
assertThatLogInWasSuccesfull	LoginHelper	Ассерт, що підтверджує вдалий вхід
4. Клас MainHelper		
open	MainHelper	Метод відкриття головної сторінки
goToAdvancedSearch	AdvancedSearchHelper	Метод, що запускає розширений пошук
searchItem	SearchResultPageHelper	Метод для пошуку товару
5. Клас AdvancedSearchHelper		
inputDataAboutGood	AdvancedSearchHelper	Метод для вводу пошукового запиту
clickOnBuyItNowButton	AdvancedSearchHelper	Метод для натискання кнопки покупки
clickOnSearchButton	SearchResultPageHelper	Метод для натискання кнопки пошуку
6. Клас SearchResultPageHelper		
getSearchFieldValue	SearchResultPageHelper	Метод для отримання даних з поля пошуку
findNecessaryGoodForAssert	SearchResultPageHelper	Ассерт, що знайдений товар має пошукову інформацію в назві
clickOnSearchedGood	ItemHelper	Метод для переходу на сторінку інформації про товар

Окремо від допоміжних класів стоять класи тестових сценаріїв. На відміну від сторінок і хелперів ці класи побудовані без дотримання принципів об’єктно-орієнтованого програмування. Тобто об’єкти цих класів ніде не створюються і не використовуються

В даному тестовому фреймворці кожен клас тестових сценаріїв відповідає одному тестовому набору. Тому в кожному класі зберігаються константні поля з інформацією про тестовий набір, методи тестів (методи з анотацією @Test), методи для збереження інформації до бази даних та (методи з анотаціями @BeforeSuit, @BeforeTest, @AfterSuit, @AfterTest). Всі ці методи мають тип значення, що повертається void та не приймають на вхід жодних даних.

Тестові дані подаються на вхід тестовим методам за допомогою антації @Test і її атрибутів dataProvider та dataProviderClass. Перший атрибут отримує набір тестових даних, що згруповано і збережено в окремому класі. Другий атрибут ідентифікує клас, в якому знаходяться тестові дані.

Діаграма класів має на меті демонстрацію того, як працює зображений на ній програмний продукт поза очами користувача. Всі зв'язки між класами, виклики хелперів, знаходження елементів веб-сторінок за допомогою еквівалентів у вигляді java-класів залишаються всередині фреймворку автоматизованого тестування.

Проте не можна сказати, що внутрішня діяльність програмного продукту залишається в таємниці від користувача. Оскільки графічний інтерфейс користувача являється ланкою, що зв'язує функціонал продукту і користувача, то саме завдяки змінам в інтерфейсі оператор продукту має змогу отримати інформацію про перебіг поточного процесу.

Оскільки елементи інтерфейсу здатні до перебування в різних станах, між якими відбувається перехід в процесі взаємодії з користувачем, то зручно описати множину станів та подій-переходів за допомогою діаграми станів.

Діаграма станів елементів графічного інтерфейсу користувача наведена на рис. 3.4.

3.4 Тестування програмної системи

Тестування будь-якого програмного продукту – процес довгий та всебічний. Одна із семи головних парадигм тестування свідчить про те, що вичерпне тестування програмного продукту неможливе. Проте будь-який програмний продукт повинен бути протестований. Навіть якщо його основною функціональністю є автоматизація процесу тестування.

З урахуванням того, що існує значна кількість видів і типів тестування, в умовах обмеженого часу та специфічного призначення фреймворку, необхідно виділити лише ті основні типи тестування, які забезпечать покриття вимог та не призведуть до надлишковості.

В першу чергу необхідно розглянути рівні тестування: модульне, інтеграційне та системне тестування. Кожен з цих рівнів направлено на перевірку правильного функціонування окремих модулів (модульне тестування), взаємодії модулів (інтеграційне тестування) та системи в цілому (системне). Беззаперечною є користь подібного підходу для великих програмних продуктів, модулі яких виконують атомарні функції і інтеграція цих модулів спричиняє приріст функціональності.

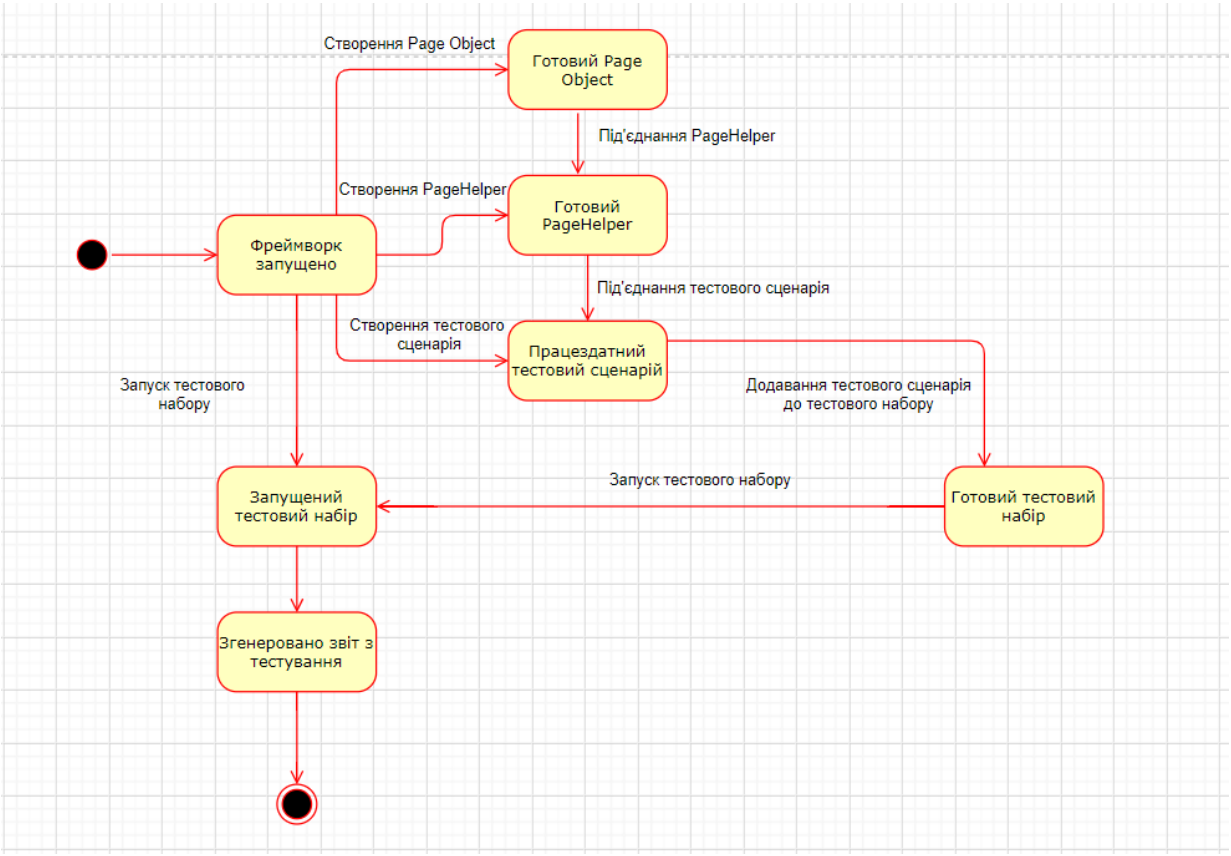

Рисунок 3.4– Діаграма станів елементів графічного інтерфейсу

Натомість у випадку фреймворку автоматизованого тестування кожна його складова не може існувати без інших. Всі його частини слугують для забезпечення одного функціоналу. Тому поділ на рівні тестування в даному випадку не має великого сенсу, адже всі тести фактично підпадають під поняття системних тестів.

Іншим способом класифікації видів тестування є поділ на функціональне та нефункціональне. Очевидно, що функціональне тестування є унікальним і обов'язковим для будь-якого програмного продукту, адже під час цього процесу тестування перевіряється функціонал. Натомість нефункціональне тестування має в собі велику кількість підвидів серед яких можна виділити наступні: тестування

продуктивності, тестування безпеки, тестування зручності, тестування установки тощо.

Тестування продуктивності має на меті перевірку працездатності програмного продукту в умовах різноманітних навантажень. Оскільки цільовою аудиторією даного фреймворку є команда тестувальників, яка є доволі обмеженою, то значної потреби в даному виді тестування нема.

Тестування безпеки має на меті перевірку механізмів захисту персональних даних, які використовуються в процесі роботи. Як і попередній вид тестування, тестування безпеки є теж неактуальним для даного продукту, адже процес авторизації не передбачений і нема даних, що потребують захисту.

Тестування зручності та тестування установки також є надмірними. Оскільки тестовий фреймворк має дуже обмежений інтерфейс, то імплементація тестування зручності має високу трудність і низький пріоритет. Тестування установки повинно передбачати потенційні проблеми інсталяції, проте оскільки для розгортування фреймворку автоматизованого тестування необхідно лише IDE з підтримкою мови Java, створення тестових вимог для інсталяційного тестування також є невиправданим.

Виходячи з усього вищесказаного можна зробити висновок, що для тестування фреймворку автоматизованого тестування є сенс створення тест-вимог та тест-планів для функціонального тестування.

Тест-вимоги для функціональних тестів фреймворку автоматизованого тестування наведені в табл. 3.5.

Таблиця 3.5 – Тест-вимоги для функціональних тестів

Ідентифікатор	Назва тест-вимоги	Опис тест-вимоги
1	2	3
Т-1	Створення нового тестового сценарія	Перевірити, що при внесенні скрипту для тестового сценарію з анотацією @Test з’являється можливість запуску тестового сценарію
Т-2.	Додавання нових тестових даних	Перевірити, що при створенні нового файлу з тестовими даними, дані можуть бути викликані за допомогою атрибутів анотації @Test
Т-3.	Актуалізація тестового сценарію	Перевірити, що при внесенні змін в скрипт тестового сценарію залишається можливість запуску тестового сценарію
Т-4.	Внесення змін в тестові набори	Перевірити, що при зміні кількості тестових сценаріїв в тестовому наборі, запуск тестового набору відбудеться без змін

Т-5.	Формування звіту проходження тестового набору	Перевірити, що після завершення виконання тестового набору і виконання команди Site буде згенеровано звіт про проходження останнього тестового набору

Закінчення таблиці 3.5

1	2	3
Т-6.	Внесення змін в тестові дані	Перевірити, що при внесенні змін в тестові дані під час виконання відповідного тестового сценарію будуть використані нові тестові дані
Т-7.	Запуск тестових наборів	Перевірити, що при запуску тестового набору через xml-файл всі тестові сценарії, що входять в тестовий набір будуть запущені
Т-8.	Збереження інформації про запуск тестових наборі до бази даних	Перевірити, що після завершення виконання сценаріїв із тестового набору в базі даних буде збережено інформацію про тестові сценарії, набори та останню сесію
Т-9.	Створення нового тестового набору	Перевірити, що після формування xml-файлу з посиланнями на тестові сценарії під час запуску тестового набору будуть запущені всі тестові сценарії

Тест-вимоги, створені на базі існуючих вимог (в даному випадку функціональних) є основою для процесу перевірки працездатності фреймворку. Проте хоча тест-вимога є необхідним елементом в процесі тестування, вона не є елементом достатнім для старту цього процесу.

Оскільки тест-вимога зазвичай відповідає структурі розділу функціональних вимог до системи, то вона фактично лише описує ту функціональність, яку треба перевірити. Натомість механізм і техніка виконання такої перевірки описується вже в тест-плані.

Далі наведено тест-плани, відповідно до існуючих тест-вимог.

Тестовий приклад: № 1

Призначення: Перевірити, що при внесенні скрипту для тестового сценарію з анотацією @Test з’являється можливість запуску тестового сценарію (табл. 3.6)

Тест-вимога, що перевіряється: Т-1.

Передумови для тесту: фреймворк має бути запущений, створена сторінка з наявним елементом, в класі хелпер існує методі існує метод для перевірки існування методу.

Критерій проходження тесту: всі реальні значення збігаються з очікуваними.

Таблиця 3.6 – Тестовий приклад №1

№	Крок сценарію	Очікуваний результат	Отриманий результат	Відмітка про проходження кроку сценарію
1.	В тестовому класі створити метод firstTest()	Метод має бути відображено на екрані	Метод відображено на екрані	Так
2.	Створити об’єкт PageHelper та викликати метод assertTest()	Дані мають з’явитись на екрані	Дані з’явились на екрані	Так
3.	Додати анотацію @Test над тестовим методом	Повинна з’явитись кнопка запуску тестового сценарію	Кнопка з’явилась	Так

Відмітка про проходження тесту (пройдений/не пройдений): пройдений.

Тестовий приклад: № 2

Призначення: Перевірити, що при створенні нового файлу з тестовими даними, дані можна викликати за допомогою атрибутів анотації @Test (табл. 3.7)

Тест-вимога, що перевіряється: Т-2.

Передумови для тесту: фреймворк має бути запущений, існує тестовий сценарій для перевірки логіну, дані про пошту та пароль знаходяться в скрипті тестового сценарію

Критерій проходження тесту: всі реальні значення збігаються з очікуваними.

Таблиця 3.7 – Тестовий приклад №2

№	Крок сценарію	Очікуваний результат	Отриманий результат	Відмітка про проходження кроку сценарію
1	2	3	4	5

1.	В тестовому сценарії видалити дані про пошту та пароль, що передаються для введення	Повідомлення про відсутність тестових даних повинно з’явитись	Повідомлення про відсутність тестових даних з’являється	Так

Продовження таблиці 3.7

1	2	3	4	5
2.	Створити клас Credentials, в якому записати дані про пошту та пароль в форматі String та анотацією @DataProvider(name = "valid_credentials")	Файл має існувати	Файл існує	Так
3.	Додати в анотацію @Test атрибути dataProviderClass = Credentials.class і dataProvider = "valid_credentials"	Кнопка запуску сценарію має бути активна, повідомлень про помилки не повинно бути	Кнопка активна, повідомлень немає	Так

Відмітка про проходження тесту (пройдений/не пройдений): пройдений.

Тестовий приклад: № 3

Призначення: Перевірити, що при внесенні змін в скрипт тестового сценарію залишається можливість запуску тестового сценарію. (табл. 3.8)

Тест-вимога, що перевіряється: Т-3.

Передумови для тесту: фреймворк має бути запущений, існує тестовий сценарій для перевірки логіну

Критерій проходження тесту: всі реальні значення збігаються з очікуваними.

Таблиця 3.8 – Тестовий приклад №3

№	Крок сценарію	Очікуваний	Отриманий	Відмітка про

		результат	результат	проходження кроку сценарію
1	2	3	4	5
1.	В тестовому сценарії замінити ланцюг викликів на почерговий виклик методів	Кнопка запуску сценарію має бути активна, повідомлень про помилки немає	Кнопка активна, повідомлень немає	Так

Закінчення таблиці 3.8

1	2	3	4	5
2.	Запустити тестовий сценарій	Тестовий сценарій повинен стартувати	Тестовий сценарій стартував	Так

Відмітка про проходження тесту (пройдений/не пройдений): пройдений.

Тестовий приклад: № 4

Призначення: Перевірити, що при зміні кількості тестових сценаріїв в тестовому наборі, запуск тестового набору відбудеться без змін. (табл. 3.9)

Тест-вимога, що перевіряється: Т-4.

Передумови для тесту: фреймворк має бути запущений, існує тестовий сценарій для перевірки логіну і тестовий набір, що має цей сценарій.

Критерій проходження тесту: всі реальні значення збігаються з очікуваними.

Таблиця 3.9 – Тестовий приклад №4

№	Крок сценарію	Очікуваний результат	Отриманий результат	Відмітка про проходження кроку сценарію
1	2	3	4	5
1.	В тестовому класі скопіювати сценарій для перевірки логіну, замінивши назву на loginWithWrongData і замінити вхідні дані на наступні: адреса електронної пошти «vrongmeil@gmeil.net» та	Кнопка запуску сценарію має бути активна, повідомлень про помилки немає	Кнопка активна, повідомлень немає	Так

	паролем «123456»			
2.	Додати тестовий сценарій в список сценаріїв в xml-файлі	Тестовий сценарій повинен відобразитись в файлі	Тестовий сценарій відображено	Так
3.	Запустити тестовий набір	В тестовому наборі має бути присутній тест loginWithWrongData	Тест присутній	Так

Відмітка про проходження тесту (пройдений/не пройдений): пройдений.

Тестовий приклад: № 5

Призначення: Перевірити, що після завершення виконання тестового набору і виконання команди Site буде згенеровано звіт про проходження останнього тестового набору. (табл. 3.10)

Тест-вимога, що перевіряється: Т-5.

Передумови для тесту: фреймворк має бути запущений, існує тестовий сценарій для перевірки логіну і тестовий набір, що має цей сценарій.

Критерій проходження тесту: всі реальні значення збігаються з очікуваними.

Таблиця 3.10 – Тестовий приклад № 5

№	Крок сценарію	Очікуваний результат	Отриманий результат	Відмітка про проходження кроку сценарію
1	2	3	4	5
1.	Запустити тестовий набір	Інформація про проходження тестового набору повинна відобразитись	Інформація відобразилась	Так
2.	В меню Maven натиснути команду Site	В папці target/allure-report має з’явитись файл index.html	Файл з’явився	Так

Відмітка про проходження тесту (пройдений/не пройдений): пройдений.

Тестовий приклад: № 6

Призначення: Перевірити, що при внесенні змін в тестові дані під час виконання відповідного тестового сценарію будуть використані нові тестові дані.

Тест-вимога, що перевіряється: Т-6. (табл. 3.11)

Передумови для тесту: фреймворк має бути запущений, існує тестовий сценарій для перевірки логіну.

Критерій проходження тесту: всі реальні значення збігаються з очікуваними.

Таблиця 3.11 – Тестовий приклад №6

№	Крок сценарію	Очікуваний результат	Отриманий результат	Відмітка про проходження кроку сценарію
1	2	3	4	5
1.	Замінити пошту з «Stanislav.Harkavyi@hneu.net» на «Stanislav.Harkavyi@gmail.com»	Нова інформація повинна з’явитись	Інформація з’явилась	Так
2.	Запустити тестовий сценарій в режимі Debug і дійти до етапу вводу даних	В поле пошти повинно бути введено нові дані	Нові дані введено	Так

Відмітка про проходження тесту (пройдений/не пройдений): пройдений.

Тестовий приклад: № 7

Призначення: Перевірити, що при запуску тестового набору через xml-файл всі тестові сценарії, що входять в тестовий набір будуть запущені. (табл. 3.12)

Тест-вимога, що перевіряється: Т-7.

Передумови для тесту: фреймворк має бути запущений, існує тестовий набір з тестовими сценаріями.

Критерій проходження тесту: всі реальні значення збігаються з очікуваними.

Таблиця 3.12 – Тестовий приклад № 7

№	Крок сценарію	Очікуваний результат	Отриманий результат	Відмітка про проходження кроку сценарію
1	2	3	4	5
1.	Запустити тестовий набір	Інформація про запущені тести має відобразитись	Інформація відобразилась	Так
2.	Відкрити xml-файл і порівняти список тестів, відображених в кроці 1 і список тестів в файлі	Списки тестів повинні співпадати	Списки тестів співпадають	Так

Відмітка про проходження тесту (пройдений/не пройдений): пройдений.

Тестовий приклад: № 8

Призначення: Перевірити, що після завершення виконання сценаріїв із тестового набору в базі даних буде збережено інформацію про тестові сценарії, набори та останню сесію. (табл. 3.13)

Тест-вимога, що перевіряється: Т-8.

Передумови для тесту: фреймворк має бути запущений, існує тестовий набір з тестовими сценаріями, в базі даних не повинно бути інформації про тестовий набір та сценарії, що в нього входять.

Критерій проходження тесту: всі реальні значення збігаються з очікуваними.

Таблиця 3.13 – Тестовий приклад № 8

№	Крок сценарію	Очікуваний результат	Отриманий результат	Відмітка про проходження кроку сценарію
1	2	3	4	5
1.	Запустити тестовий набір	Інформація про запущені тести повинна відобразитись	Інформація відобразилась	Так
2.	Відкрити таблиці Test, Test Suit та Test Session в базі даних	Інформація про тестовий сценарій, тестовий набір та тестову сесію	Інформація внесена	Так

		повинна бути внесена		

Відмітка про проходження тесту (пройдений/не пройдений): пройдений.

Тестовий приклад: № 9

Призначення: Перевірити, що після формування xml-файлу з посиланнями на тестові сценарії під час запуску тестового набору будуть запущені всі тестові сценарії. (табл. 3.14)

Тест-вимога, що перевіряється: Т-9.

Передумови для тесту: фреймворк має бути запущений, тестові сценарії повинні існувати.

Критерій проходження тесту: всі реальні значення збігаються з очікуваними.

Таблиця 3.14 – Тестовий приклад № 9

№	Крок сценарію	Очікуваний результат	Отриманий результат	Відмітка про проходження кроку сценарію
1	2	3	4	5
1.	Створити xml-файл, в який внести в тег classes такі дані <class name="search.SearchFieldTest"/>	Дані повинні з’явитись	Дані з’явились	Так
2.	Запустити тестовий набір з xml-файлу, створеного на кроці 1	Інформація про запущені тести повинна з’явитись	Інформація з’явилась	Так
3.	Відкрити клас SearchFieldTest і порівняти список тестів в класі зі списком тестів, відображених під час кроку 2	Списки тестів повинні співпадати	Списки тестів співпадають	Так

Відмітка про проходження тесту (пройдений/не пройдений): пройдений.

На основі проведеного тестування функціональних вимог було проведено дев’ять тестів, з яких дев’ять отримали відмітку «пройдений». На основі цього

можна зробити висновок, що поточний відсоток проходження тестів дорівнює ста і якість розробленого програмного продукту є задовільною.

3.5 Розгортання програмного продукту

Поширення фреймворку автоматизованого тестування значно легше ніж інсталяція більшості настільних додатків. Для початку роботи необхідно мати сам проект фреймворку, а також IDE Intellij IDEA.

Фреймворк може бути запущеним на будь-якому комп’ютері незалежно від операційної системи та технічних характеристик. Стосовно вимог до програмних засобів основною вимогою окрім наявності IDE є використання JDK з підтримкою Java 8.

Дана версія не є останньою і тому використання більш пізніх версій може призвести до помилок. Проте саме ця версія є одною з найбільш розповсюджених, стабільних і функціонально повних, тому рекомендовано використовувати саме Java 8.

Окрім JDK необхідним плагіном, який необхідно встановити в IDE самостійно перед початком роботи є плагін Lombok. Даний плагін спрощує роботу з анотаціями та сприяє покращенню зрозумілості тестових сценаріїв.

Стосовно самого процесу інсталяції, то він є дуже простим.

В першу чергу необхідно мати актуальну версію проекту фремворку. Для цього можна завантажити його за допомогою github. Після завантаження проекту необхідно відкрити його за допомогою IDE. Система може запропонувати оновити поля Dependencies для Apache Maven. Для початку роботи треба дозволити оновлення і дочекатись доки Apache Maven завантажить необхідні бібліотеки з віддаленого репозиторію.

ВИСНОВКИ

Метою даного проекту було створення фреймворку автоматизованого тестування, який би забезпечував контроль якості програмного забезпечення. Розроблений продукт дозволяє скоротити затрати як часу, так і людських ресурсів на провадження процесу тестування. Даний факт однозначно сприяє підвищенню ефективності компанії, що використовує фреймворк в процесі тестування.

Результатом першого етапу стало розуміння структури процесу тестування програмного забезпечення, а також типові проблеми, з якими стикаються підприємства під час використання ручного підходу до тестування. Також було проаналізовано наявні на ринку інструменти для автоматизації тестування і на основі їх було сформовано загальне уявлення про функціональність фреймворку автоматизованого тестування.

На другому етапі загальні поняття було сформовано в специфікацію вимог до програмного продукту. На цьому етапі було створено глосарій предметної області, розроблено варіанти використання, сформульовано специфікації функціональних та нефункціональних вимог, а також спроектовано інтерфейс користувача.

На заключному етапі було обрано та впроваджено відповідні технічні рішення. Зокрема це вибір і впровадження СКБД, моделювання бази даних, проектування і розробка самого програмного продукту за допомогою спеціалізованого інструментарію, а також процес перевірки якості розробленого фреймворку.

Результатом розробки є фреймворк автоматизованого тестування, що дозволяє провадити роботу з тестовими сценаріями та наборами за допомогою інструментів TestNG та Selenide. Окрім цього впроваджено засоби збереження інформації про тестові сесії в базі даних та генерація звіту за допомогою інструменту Allure.

Подальші варіанти розвитку програмного продукту можуть включати в себе розширення наявного функціоналу та впровадження нового. Серед нового функціоналу можна відмітити підтримку нових браузерів, розробка додаткових універсальних тестових кроків, оптимізація та прискорення роботи, розробка системи аналізу причин виявлених дефектів.

Даний програмний продукт є готовим до використання. Завдяки інтуїтивно зрозумілим інструментам, що забезпечують низький поріг входження, даний програмний продукт може однаково ефективно використовуватись як командою спеціалістів-тестувальників, так і окремими розробниками-фрілансерами.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Автоматизация тестирования: выбор инструмента. // OpenQuality.ru. Качество программного обеспечения. URL: http://blog.openquality.ru/tool-choice
2. Бек, К. Шаблоны реализации корпоративных приложений / К. Бек., Москва : Вильямс, 2017.
3. Бєрнс Д. Selenium 2 засоби тестування: керівництво для початківців / Д. Бєрнс. Packt Publishing, 2012.
4. Блек Р. Ключові процеси тестування / Р. Блек., Лорі, 2006. 544 с.
5. Буч, Г. Язык UML. Руководство пользователя: Пер. с англ. Мухин Н. / Г. Буч, Д. Рамбо, А. Якобсон М. : ДМК Пресс, 2006., 496 с.
6. Введение в BDD [Електронний ресурс]. URL: http://agilerussia.ru/practices/introducing-bdd/
7. Калбертсон Р., Браун К., Кобб Г.. Быстрое тестирование, 2004. 379 с.
8. Канер, С. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений / Пер. с англ. С. Канер, Д. Фолк, Е. К. Нгуен., Киев. : ДиаСофт, 2001., 544 с
9. Коваленко Д. Selenium Design Patterns and Best Practices / за ред. Д. Коваленко PACKT Publishing, 2014.
10. Куликов С. Тестирование программного обеспечения. Базовый курс. Минск: Четыре четверти, 2015.
11. Курняван Б. Програмування web-додатків на мові Java / за ред. Б. Курняван Лорі, 2009.
12. Макконнелл, С. Совершенный код. Мастер класс. Пер. с англ. / за ред. С. Макконнелл., М. : Русская редакция, 2010, 896 с.
13. Осипенко, Н. Б. Стандартизация и сертификация программного обеспечения. Тексты лекций. / Н. Б. Осипенко., Министерство образования РБ, Гомельский гос. ун-т им. Ф. Скорины., Гомель: ГГУ им. Ф. Скорины, 2012.
14. Метрики в тестировании. URL: http://blog.shumoos.com/archives/271.
15. Пирамида автоматизации и другие геометрические фигуры, AT.Info URL: http://automated-testing.info/
16. Портал Про Тестинг – Тестирование Программного Обеспечения, URL: http://www.protesting.ru.
17. Тампе, Л. Введение в тестирование программного обеспечения / за ред. Л. Тампе., М. : Вильямс, 2003., 368 с.
18. Тестирование и качество ПО. URL: http://software-testing.ru/
19. Фаулер М. Шаблоны корпоративных приложений, Москва: Вильямс, 2016.

20. Copland L. A Practitioner's Guide to Software Test Design., London: STQE Publishing, 2004.

21. Deitel P. Java How to Program / P. Deitel, H. Deitel., Prentice Hall, 2015.

22. Duvall Paul, Matyas Stephen M., Glover Andrew. Continuous Integration: Improving Software Quality and Reducing Risk (The Addison-Wesley Signature Series). Addison-Wesley Professional, 2007. ISBN: 0321336380.

23. Fewster Mark. Software Test Automation: Effective Use of Test Execution Tools/ Mark Fewster, Dorothy Graham. – Addison-Wesley & Son, Ltd, 2000., 574 p.

24. Freeman Elisabeth, Freeman Eric, Bates Bert, Sierra Kathy. Head First Design Patterns, O' Reilly & Associates, Inc., 2004., ISBN: 0596007124.

25. Graham Dorothy. ROI of test automation:benefit and cost, 2010. URL: http://www.dorothygraham.co.uk/downloads/generalPdfs/ProfTesterROI.pdf.

26. Guide to the Software Engineering Body of Knowledge: SWEBOK, Version 3.0, IEEE; 2013

27. Hayes Linda G.. Automated Testing Handbook., Software Testing Inst; 2nd edition, 2004., 182 p.

28. ltd Cucumber. Cucumber. Simple, human collaboration. URL: https://cucumber.io.

29. Modern Test Case Management Software for QA and Development Teams. URL: http://www.gurock.com/testrail/

30. Myers Glenford J. The Art of Software Testing /, Revised and Updated by Tom Badgett, Todd M.Thomas, Corey Sandler. 2nd ed., Hoboken, New Jersey.: John Wiley & Sons, Inc., 2004

31. Osherove Roy. The Art of Unit Testing: With Examples in .Net. 1st edition. Greenwich, CT, USA : Manning Publications Co., 2009. ISBN: 1933988274, 9781933988276.

32. PageObject. Martin Fowler. URL: https://martinfowler.com/bliki/PageObject.html.

33. Test Design Considerations // Selenium HQ. Browser automation. URL: https://www.selenium.dev/documentation/en/

34. TIOBE Index. URL: https://tiobe.com/tiobe-index/.