

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіоелектроніки,
Факультет комп'ютерних наук і технологій

(повне найменування інституту, назва факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ МЕТОДІВ ТА ПРОГРАМНИХ ЗАСОБІВ
АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ САЙТІВ НА КОМП'ЮТЕРАХ І
МОБІЛЬНИХ ПРИСТРОЯХ. RESEARCH OF METHODS AND SOFTWARE
TOOLS FOR AUTOMATED SITE TESTING ON COMPUTERS AND MOBILE
DEVICES

Виконав: студент(ка) 2 курсу, групи КНТ-210м
Спеціальності

122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Системи штучного інтелекту

Тимофєєв Є. Г.

(прізвище та ініціали)

Керівник Субботін С. О.

(прізвище та ініціали)

Рецензент Гофман Є. О.

(прізвище та ініціали)

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

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

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
 “ ” 20__ року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТА(КИ)

Тимофєєва Єгора Григоровича
 (прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Дослідження методів та програмних засобів автоматизації тестування сайтів на комп'ютерах і мобільних пристроях.
Research of methods and software for automating site testing on computers and mobile devices
- керівник проекту (роботи) Субботін Сергій Олександрович,
 (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
- затверджені наказом закладу вищої освіти від “5” листопада 2021 року № 435
2. Строк подання студентом проекту (роботи) 13 грудня
3. Вихідні дані до проекту (роботи) рекомендована література, технічне завдання
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Методологічна основа роботи. Поняття тестування у процесі розробки. 2. Аналіз інструментів автоматизованого тестування та обробки результатів. 3. Методології та технології реалізації програмного продукту. 4. Організаційна частина. 5. Реалізація системи.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
Основна частина	Субботін С. О., професор		
Нормконтроль	Камінська Ж. К., асистент		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Постановка завдання роботи.	1-2 тижні	Завдання, ТЗ
2	Методологічна основа роботи. Поняття тестування у процесі розробки.	3-5 тижні	Розділ 1
3	Аналіз інструментів автоматизованого тестування та обробки результатів.	6-7 тижні	Розділ 2
4	Методології та технології реалізації програмного продукту.	8-10 тижні	Розділ 3
5	Реалізація системи.	11-13 тижні	Розділ 4-5
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	14-15 тижні	Додатки
7	Захист роботи.	16 тиждень	

Студент (ка) _____ Тимофєєв Є. Г.
(підпис) (прізвище та ініціали)Керівник проекту (роботи) _____ Субботін С. О.
(підпис) (прізвище та ініціали)

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК.....	8
ВСТУП.....	9
1 МЕТОДОЛОГІЧНА ОСНОВА РОБОТИ. ПОНЯТТЯ ТЕСТУВАННЯ У ПРОЦЕСІ РОЗРОБКИ.....	11
1.1 Поняття тестування, його цілі та головні завдання	11
1.2 Види технік тестування.....	13
1.3 Підходи до виконання тестування	18
1.4 Висновки за розділом	23
2 АНАЛІЗ ІНСТРУМЕНТІВ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ТА ОБРОБКИ РЕЗУЛЬТАТІВ.....	24
2.1 Фреймворки для тестування	24
2.2 Інструменти автоматизації для комп'ютерів	38
2.3 Інструменти автоматизації для мобільних пристроїв	45
2.4 Системи звітності тестування.....	51
2.5 Висновки до розділу	53
3 МЕТОДОЛОГІЇ ТА ТЕХНОЛОГІЇ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ПРОДУКТУ	55
3.1 Мова програмування	55
3.2 Фреймворк тестування	57
3.3 Сервер, який дозволяє тестам використовувати екземпляри веббраузера, запущені на віддалених машинах	59
3.4 Звітність	60
3.5 Хмарна платформа для веб- та мобільного тестування.....	60
3.6 Інструмент автоматизації для запуску скриптів і тестування мобільних вебзастосунків	62
3.7 Система автоматичного збирання.....	63
3.8 Інструмент для автоматизації роботи в веб-браузері.....	63
3.9 Інструмент для безперервної інтеграції	64

3.10 Середовище реалізації розроблюваного програмного продукту.....	65
3.11 Висновки до розділу	67
4 ОРГАНІЗАЦІЙНА ЧАСТИНА.....	68
4.1 Загальні положення	68
4.2 Основний процес, який повинен виконувати програмний продукт.....	68
4.3 Загальна модель роботи системи та ролі користувачів системи	69
4.4 Взаємодія користувача з фреймворком.....	72
5 РЕАЛІЗАЦІЯ СИСТЕМИ	75
5.1 Основні класи.....	75
5.2 Допоміжні класи	77
5.3 Клас для логування.....	79
ВИСНОВКИ	80
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ	82
Додаток А Текст програми	85
Додаток Б Презентація	114

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи магістра: 121 с., 4 табл., 22 рис., 2 дод., 32 джерела.

АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ, JAVA, INTELLIJ IDEA, ІНСТРУМЕНТИ ТЕСТУВАННЯ, ЦИКЛ РОЗРОБКИ, TESTNG, APPIUM, SELENIUM, GRADLE, ALLURE, BROWSERSTACK, SELENIUM GRID, SELENIUM WEBDRIVER.

Об'єкт дослідження – процес тестування програмних продуктів.

Предмет дослідження – засоби та методи автоматизації процесу тестування програмних продуктів.

Метою роботи є створення фреймворку для автоматизації процесу тестування.

Інструмент розробки – IntelliJ IDEA.

При роботі над кваліфікаційною роботою було виконано порівняльний аналіз різних підходів, методів, фреймворків та інструментів для тестування; відповідно до аналізу було обрано найкращі з них, а також створено власний фреймворк для автоматизації процесу тестування.

Результати роботи можна використовувати у процесі розробки програмного продукту, а саме на етапі тестування.

ABSTRACT

Explanatory note to the master's thesis: 121 pages, 22 figures, 4 table, 2 appendixes, 32 sources.

AUTOMATED TESTING, JAVA, INTELLIJ IDEA, TESTING TOOLS, DEVELOPMENT CYCLE, TESTNG, APPIUM, SELENIUM, GRADLE, ALLURE, BROWSERSTACK, SELENIUM GRID, SELENIUM WEBDRIVER.

The object of research is the process of testing software products.

The subject of research - means and methods of automating the process of testing software products.

The aim of the work is to create a framework for automating the testing process.

Development tool - IntelliJ IDEA.

During the work on the qualification work, a comparative analysis of different approaches, methods, frameworks and tools for testing was performed; according to the analysis, the best of them were selected, and their own framework was created to automate the testing process.

The results of the work can be used in the process of software development, namely at the testing stage.

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface, прикладний програмний інтерфейс;

ATDD – Acceptance Test-Driven Development, розробка на основі приймального тестування;

BDD – Behavior-Driven Development, розробка на основі поведінки;

CI – Continuous Integration, неперервна інтеграція;

CPU – Central processing unit, центральний процесор;

GB – Gigabyte, гігабайт;

GUI – Graphical user interface, графічний інтерфейс користувача;

HTML – HyperText Markup Language, мова розмітки гіпертексту;

HTTP – HyperText Transfer Protocol, протокол передачі гіпертекстових документів;

IDE – Integrated development environment; інтегроване середовище розробки;

JSON – JavaScript Object Notation, запис об'єктів JavaScript;

MB – Megabyte, мегабайт;

OS – Operation System, операційна система;

RAM – Random Access Memory, операційна пам'ять;

SDK – Software Development Kit, набір засобів розробки;

SSD – Solid-State Drive, твердотілий накопичувач;

TDD – Test-Driven Development, розробка на основі тестування;

UI – User interface, дизайн інтерфейсу користувача;

XML – eXtensible Markup Language, розширювана мова розмітки;

ПП – програмний продукт;

ТЗ – технічне завдання.

ВСТУП

В сучасному світі дуже великий вплив на практично всі сфери життя людини має програмне забезпечення, а саме його використання для роботи персональних стаціонарних комп'ютерів, мобільних телефонів, планшетів, ноутбуків тощо. Очевидно, що з кожним роком зростає складність розробки, а також підтримка роботи програмних продуктів, а отже, виникає гостра необхідність у скороченні часу та коштів, витрачених на реалізацію та подальшу підтримку роботи програмних продуктів. Варто зазначити, що уже зараз різноманітні структури, у тому числі державні, сфери бізнесу та підприємництва намагаються полегшити роботу та зекономити час користувачів, витрачений на неї саме за допомогою різноманітних програмних продуктів. Насправді, далеко не кожен користувач може оцінити лише за інтерфейсом, наскільки буває ресурсозатратною та важкою розробка та тестування будь-якого застосунку.

Незважаючи на це, вимоги до розробників стають більш жорсткими з року в рік. Це стосується не лише якості фінального продукту, а й строків його реалізації. У світі існують безліч компаній, кількість працівників, яких налічують десятки тисяч працівників і всі вони працюють над розробкою єдиного програмного продукту, а також над його тестування та покращенням. Чи не найважливішим для компаній є саме етап тестування. Якщо помилка у розроблюваному програмному продукті буде знайдена ще до його появи в загальному доступі, значно знизиться вартість її виправлення, адже необхідно буде лише виправити необхідний фрагмент коду, що призвів до помилки, а також обов'язково повторно протестувати. А ось якщо саме користувач знаходить якусь помилку у програмному продукті, тобто фактично вразливе місце у системі, тоді уже така помилка може стати фатальною та навіть відібрати людське життя чи погіршити стан здоров'я, в залежності від сфери використання розроблюваного продукту, а вартість виправлення такої помилки буде значно вищою та більш довготривалою.

Все те, що було зазначено раніше, спонукає тестувати власний продукт, тобто ПЗ, на початковому етапі. Більше того, сучасні ІТ-компанії закладають у бюджет величезні суми, які витрачаються саме на тестування, хоча на перший погляд це здається непотрібним. Також варто зазначити, що багато таких компаній намагаються практично повністю автоматизувати цей процес, тобто застосовувати в своїй роботі в переважній більшості саме автоматизований вид тестування. Хоча, потрібно розуміти, що не всі процеси можна або ж потрібно автоматизувати. Мова йде про процеси, які необхідно перевірити лише один раз і тоді при їх ручній перевірці витрачається менше часу та коштів.

Мета й завдання роботи. Метою кваліфікаційної роботи є дослідження методів та засобів автоматизації сайтів, а також створення програми, яка буде слугувати, тестовим фреймворком, за допомогою якого можна легко почати автоматизоване тестування на комп'ютерах чи мобільних пристроях. Для досягнення цієї мети поставлено такі завдання:

- дослідити методи та програмні засоби автоматизації тестування сайтів на різних пристроях;
- класифікувати методи та програмні засоби автоматизації та виконати їх порівняльний аналіз;
- створити програмний продукт для написання автоматизованих тестів для комп'ютера та мобільних пристроїв.

Об'єктом дослідження – процес тестування програмних продуктів.

Предмет дослідження – засоби та методи автоматизації процесу тестування програмних продуктів.

Під час дослідження засобів, методів і розробки програми були виділені аспекти, які потребують покращення. В процесі створення виявлені слабкі сторони, того чи іншого засобу чи методу та обрані кращі варіанти для даної мети.

В якості інструменту створення програмного засобу було обрано IDEA – інтегроване середовище розробки (IDE) мовою програмування Java, яке є безкоштовним, вільно поширюваним, з відкритим вихідним кодом.

1 МЕТОДОЛОГІЧНА ОСНОВА РОБОТИ. ПОНЯТТЯ ТЕСТУВАННЯ У ПРОЦЕСІ РОЗРОБКИ

Цей розділ присвячений розгляду основних понять, які використовуються в тестуванні, а також його видів і, головне, цілі його використання загалом.

1.1 Поняття тестування, його цілі та головні завдання

За означенням, тестування програмного забезпечення — це процес технічного дослідження, який призначений для виявлення інформації про якість продукту відносно контексту, в якому він має використовуватись [1].

Іншими словами, тестування - це процес спостереження за роботою програмного продукту за різних умов, що створюються штучно, імітуючи справжнє середовище [2]. Тож з означення можна зробити висновок, що для тестування можна обрати різні галузі, зокрема, продуктивність, захищеність, зручність програмного продукту тощо.

В процесі тестування необхідно відповісти лише на одне питання – чи поведінка (або ж відповідь) програмного продукту в умовах, створених тестом, така ж, як і очікувана? Зрозуміло, що при негативній відповіді на таке запитання необхідно змінювати програмний код, усуваючи помилку.

Для процесу тестування можна виділити такі основні задачі:

- виявлення помилки, що спонукає оперативно її усунути;
- перевірка відповідності програмного продукту вимогам, сформованим в технічному завданні;
- оцінка якості роботи розробників;
- отримання інформації, яка впливає на подальший процес розробки або реалізації програмного продукту.

Однозначно, тестування необхідне тому, що при написанні програмного коду неможливо передбачити усі можливі помилки, окрім того, тестування виключає й фактор неуважності в розробників. Наслідки від допущених

помилки можуть бути різноманітними, деякі з них і справді незначні, а деякі можуть призвести до катастрофи. Тому кожен програмний продукт, без жодних виключень, повинен проходити етап тестування, перш ніж потрапляти у руки замовника.

Тестування програмних продуктів зазвичай потребують досить великих часових затрат, проте потрібно розуміти, що це невід'ємна частина розробки, без якого складно уявити процес створення будь-якої програми. Варто розуміти, що тестування насправді дозволяє зекономити безліч ресурсів, оскільки попереджає виникнення помилок у роботі програмних продуктів, які вже вийшли на ринок.

Цілі для виконання тестування можуть відрізнятися, залежно від етапу розробки програмного продукту. На першому етапі, тобто на етапі розробки програмного коду, тестування допомагає штучно викликати якомога більше помилок при штучно створених умовах, що допомагає локалізувати вразливі частини програмного продукту та виправити дефекти в них. На етапі здачі проекту, тестування необхідне для демонстрації правильної роботи програмного продукту в будь-яких ситуаціях. На кінцевому етапі, тобто при супроводі роботи програмного продукту, тестування необхідне для того, щоб при внесенні будь-яких змін в програмний код, або навіть зміні версій інструментів програмування, можна було виявити усі недоліки роботи та запобігти їх впливу на коректність роботи програмного продукту.

Враховуючи усе, що було зазначено раніше, можна визначити такі основні цілі тестування в цілому:

- надання інформації про якість програмного продукту кінцевому замовнику (можливо, у вигляді звітів або іншої супроводжуючої документації);
- підвищення якості роботи програмного продукту, шляхом виключення критичних ситуацій в роботі;
- запобігання появі дефектів, або іншими словами критичних ситуацій.

Остання ціль є основною для процесу тестування.

1.2 Види технік тестування

Тож на основі попереднього підрозділу, потрібно зазначити, що всі види тестування програмних продуктів, залежно від цілей та завдань в їх основі, можна розділити на такі групи [3]:

- функціональні;
- нефункціональні;
- пов'язані зі змінами.

1.2.1 Функціональні види тестування

Функціональне тестування може бути проведене на всіх рівнях тестування, зокрема, на приймальному (Acceptance testing), інтеграційному (Integration testing), системному (System testing) та на компонентному або модульному (Component/Unit testing), а в своїй основі функціональне тестування базується на взаємодії з іншими системами та на різних функціях [4]. Для функціонального тестування можна зазначити такі особливості та переваги:

- імітація використання. Основне завдання функціонального тестування – змусити програмний продукт поводитися так, як за фактичного його використання;
- уникнення критичних ситуацій. Очевидно, що чим раніше будуть виявлені помилки в роботі програмного продукту, тим менше буде проблем у майбутньому використанні системи;
- вчасність виявлення дефектів. Так як функціональне тестування використовується на всіх рівнях тестування, можливе вчасне виявлення помилок в роботі програмного продукту, а отже, й зниження ризиків.

Як уже було зазначено, функціональні види тестування розглядають зовнішню поведінку програмного продукту. Існує ряд найпопулярніших видів функціонального тестування [5]:

- функціональне тестування (Functional testing). Такий підвид тестування розглядає заздалегідь штучно визначену поведінку і ґрунтується на аналізі функцій компонентів або ж програмного продукту загалом;
- тестування безпеки (Security and Access Control Testing). Назва цієї стратегії тестування вказує повністю на своє призначення, її основне завдання – перевірка безпеки програмного продукту, а також цілісності і закритості системи, що не дасть можливості отримати незаконний доступ до даних, що носить у собі програмний продукт. Також, ця стратегія використовується для аналізу можливих ризиків відносно різноманітних вірусів;
- тестування взаємодії (Interoperability Testing). Цей підвид функціонального тестування, надає можливість здійснювати перевірку програмного продукту на взаємодію із іншими програмними компонентами. Окрім того, варто зазначити, що тестування взаємодії включає в собі й тестування сумісності (compatibility testing) і інтеграційне тестування (integration testing).

1.2.2 Нефункціональні види тестування

Нефункціональне тестування, необхідне для визначення характеристик програмного продукту, які можуть вимірюватися різними величинами, які не відносяться до дій користувача системи [6].

Нефункціональне тестування визначити такі особливості програмних продуктів:

- надійність програмного продукту. Тестування необхідне для визначення поведінки системи, яка надається за умови виникнення критичної ситуації;

- продуктивність програмного продукту. Тестування накладе різне навантаження на систему й визначає, як програмний реагує при цьому;
- зручність використання програмного продукту. У цьому випадку тестування дає змогу зрозуміти, на скільки простий та зручний користувацький інтерфейс та на скільки легкий й зрозумілий процес управління системою;
- масштабованість програмного продукту. Ця властивість полягає у перевірці вертикального та горизонтального масштабування, тобто у визначенні можливостей системи у випадку збільшення навантаження зі зміною різноманітних параметрів, зокрема, зміною кількості трафіку, транзакцій тощо;
- безпека програмного продукту. Нефункціональне тестування здатне перевіряти безпеку конфіденційних даних користувача;
- портативність програмного продукту. Очевидно, що зараз існує безліч пристроїв, на яких програмний продукт повинен показувати коректні результати роботи, саме нефункціональне програмування допомагає перевірити цю властивість програмного продукту, що тестується.

Тож, можна зробити висновок, що нефункціональне тестування призначене для перевірки роботи системи. Як і в функціональному тестуванні, в нефункціональному теж можна виділити основні підвиди:

- навантажувальне тестування (Performance and Load Testing). Такий підвид тестування здатен імітувати роботу одразу багатьох користувачів на одному ресурсі та визначати працездатність системи при таких навантаженнях;
- стресове тестування (Stress Testing). Такий підвид тестування має можливість виявити слабкі місця у роботі програмного продукту за умови перевантаження системи;
- тестування стабільності та надійності (Stability/Reliability Testing). При цьому тестуванні можлива зміна штучно створеного середовища роботи програмного продукту протягом певного часу. Це дозволяє виявляти різноманітні дефекти у роботі;

- об'ємне тестування (Volume Testing). Головною метою даного підвиду тестування є визначення продуктивності програмного продукту за умови збільшення кількості даних у базі даних;
- тестування установки (Installation testing). Ціль цього підвиду нефункціонального тестування – перевірка працездатності програмного продукту в умовах установки, оновлення, видалення або ж налаштування;
- тестування зручності користування (Usability Testing). Цей метод тестування необхідний для визначення зручності, зрозумілості та інтуїтивності при роботі із програмним продуктом, також тестування зручності користування враховує також привабливість системи для користувача;
- тестування на відмову та відновлення (Failover and Recovery Testing). Метою даного підвиду нефункціонального тестування є перевірка здатності програмного продукту до відновлення (або ж його окремих частин, тобто окремого функціоналу) у випадку виникнення будь-яких збоїв, пов'язаних з роботою програмного продукту, помилки у роботі мережі, перебоїв у роботі обладнання тощо. Тобто, фактично, даний метод дозволяє перевірити забезпечення збереження та цілісності користувацьких даних програмного продукту;
- конфігураційне тестування (Configuration Testing). Такий підвид нефункціонального тестування використовується спеціально для того, аби виконати перевірку програмного продукту, враховуючи різні конфігурації, які були заявлені в технічному завданні, або в іншій супроводжуючій документації, зокрема, на різних платформах, драйверах, при змінених конфігураціях девайсів тощо.

1.2.3 Пов'язані зі змінами види тестування

Останній вид техніки тестування – тестування, пов'язане зі змінами програмного продукту. Такий вид тестування необхідний для того, щоб надати підтвердження, що після того, як було змінено програмний код для запобігання

критичних ситуацій, програмний продукт працює коректно, тобто усі помилки та дефекти виправлені.

Нижче наведено підвиди такого типу тестування:

- димове тестування (Smoke Testing). Такий підвид тестування потрібно використовувати кожного разу, коли була створена нова версія програмного продукту, тобто було змінено програмний код системи. Таке тестування дає можливість на практиці переконатися, що критичні тести працюватимуть так, як і очікується, тобто поведінка програмного продукту буде коректною. Основна мета даного виду тестування – виявлення серйозних критичних ситуацій на ранньому етапі розробки і в випадку не проходження тесту з'являється можливість відхилення нової версії, що дозволяє значно зекономити час розробки в цілому. Для цього зазвичай використовується цикл тестів;

- регресійне тестування (Regression Testing). Такий підвид тестування, пов'язаного зі змінами програмного продукту, зазвичай проводиться тоді, коли з'являється необхідність в тому, щоб переконатися, що нові функції, які були додані до програмного продукту ніяк не вплинули на уже існуючі функції системи, а також перевірити. Варто зазначити, що такий тип тестування застосовується й для тих частин програмного продукту, які не змінювалися і уже були протестовані. Важливо, що такий тип тестування – це один із найпоширеніших в тестуванні загалом;

- тестування складання (Build Verification Test). Таке тестування призначене для того, аби підтвердити сумісність версії програмного продукту, що була випущена, та початкових умов;

- санітарне тестування (Sanity Testing). Це тестування і ще називають «тестуванням узгодженості». Основна мета цього підвиду – перевірка того, що усі необхідні функції працюють на низькому рівні. Такий тип тестування зазвичай використовується після кожного етапу розробки програмного продукту.

1.3 Підходи до виконання тестування

Як і раніше зазначалося, забезпечення якості розроблюваного програмного продукту – надзвичайно складний процес, який включає не тільки розробку, а й визначення критеріїв якості та коректності роботи програмного продукту, розробку стратегії тестування, які дозволятимуть проводити різні перевірки системи для виявлення помилок, їх опис та порівняння, що несе за собою й правильне внесення необхідних змін.

Одне з найважливіших питань, яке постає до початку роботи над проектом – як краще виконувати перевірку коректності роботи програмного продукту: вручну чи автоматизовано?

Правильної відповіді на це питання не існує, оскільки обидва підходи мають свої переваги та недоліки, а отже, обирати варто відштовхуючись від формулювання завдання проекту. На будь-якому проекті, без виключення, найважливіші три фактори: час, вартість та якість. Для того, аби проект був успішним, необхідно мінімізувати витрачені кошти та час, при цьому максимізуючи якість. І ось коли мова йде про етап тестування програмних продуктів, один із них може виявитися набагато ефективнішим, аніж інший. Варто зазначити, що досить часто найбільш вигідним для розробки якісного продукту є поєднання двох підходів. При цьому частка автоматичних та ручних тестів буде залежати від того, що вимагається в проекті, а також терміну виконання та бюджету.

Основна і очевидна відмінність цих підходів полягає у тому, що при ручному підході кожен із тест-кейсів запускається вручну, тобто без використання будь-яких допоміжних програмних засобів, а ось при автоматизованому підході до тестування запуск усіх тест-кейсів здійснюється за допомогою спеціально розроблених скриптів.

1.3.1 Поняття ручного тестування

Як уже раніше зазначалося, ручне тестування – це такий підхід до тестування програмного продукту, де тести виконуються повністю вручну тестувальником. Такий підхід використовується для того, аби виявити помилки та дефекти в роботі розроблюваної системи [7].

Ручне тестування призначене для того, аби перевіряти основні функції програмного продукту. Завдання ж тестувальника – виконати контрольні тести, не використовуючи додаткові засоби автоматизації та надати звіт.

При ознайомленні з цим підходом до тестування, було виділено такі переваги:

- через наявність людського фактору, можливо отримати результат набагато швидше та точніше;
- такий підхід значно дешевший, оскільки в його організації не потрібно вкладати додаткові ресурси для організації процесів автоматизації та на використання різноманітних інструментів;
- ручне тестування має значну перевагу, коли в програмному коді було проведено невеликі зміни, оскільки в цьому випадку в автоматизованому тестуванні необхідно змінювати деякі скрипти.

Незважаючи на значні переваги ручного тестування, в такому підході існують й недоліки:

- такий метод менш надійний через те, що проводиться людиною вручну;
- тести, що були написані вручну, не можна використати повторно, так як вони не записуються;
- за допомогою цього підходу досить важко виконувати деякі задачі при створенні програмного продукту.

Найчастіше такий вид тестування використовується у таких випадках:

- при тестуванні зручності користувацького інтерфейсу. Варто зазначити, що тестувальник самостійно повинен визначити наскільки зручний у

використанні програмний продукт, а також повинен проаналізувати і надати звіт про це;

- при інтуїтивному тестуванні теж необхідно використовувати цей підхід, адже тоді тестувальник не може використовувати методи автоматизації, оскільки, перевірка виконується лише виходячи із власного досвіду.

1.3.2 Поняття автоматизованого тестування

Для використання підходу автоматизованого тестування, тестувальники створюють спеціальні сценарії, за якими відбувається автоматизоване виконання тесту частини програмного продукту, що покриває розроблений сценарій [8]. Для цього використовуються спеціальні інструменти для автоматизації. Основна ціль такого підходу до тестування – зменшити кількість витраченого часу на перевірку фрагментів коду.

У процесі автоматизованого тестування виконується перевірка частини програмного продукту за написаним сценарієм, а отриманий результат звіряється із очікуваним.

Незважаючи на те, що всі процеси тестування виконуються автоматично, автоматизація все одно потребує того, аби на початкових етапах вручну створювалися сценарії для тестування програмного продукту.

Автоматичне тестування безумовно має безліч переваг, зокрема:

- автоматичне тестування дозволяє покрити більший фрагмент програмного коду, а також дозволяє знайти більше помилок в програмному продукті;

- процес отримання результатів набагато швидший та ефективніший, особливо не на початкових етапах, так як більшість процесу тестування уже автоматизована;

- можливо повторне використання тих самих операцій тестування, оскільки увесь процес записується;

- для автоматизованого тестування не потрібна постійна і безпосередня участь тестувальника, так як процес виконується абсолютно самостійно;
- забезпечується висока продуктивність роботи і пришвидшення процесу розробки;
- зараз існує безліч різноманітних інструментів, які сумісні із автоматизованим тестуванням;
- ймовірність того, що тестування покриватиме повністю весь код, набагато більша, оскільки при автоматизації покриваються навіть найменші фрагменти коду.

Незважаючи на величезну кількість переваг автоматизованого тестування, існує також ряд недоліків:

- такий підхід неможливо використовувати для тестування зручності користувацького інтерфейсу;
- хоч більшість засобів для автоматизації тестування є безкоштовними, але все ж існують й інструменти, які можуть значно вплинути на формування бюджету, виділеного на розробку програмного продукту;
- очевидно, що в роботі усіх інструментів автоматизації все ж можуть траплятися помилки;
- надзвичайно важливим є написання тестового сценарію, що є досить дорогим.

За допомогою автоматизованого підходу найчастіше проводяться такі види тестування [9]:

- регресійне тестування;
- тестування навантаження;
- тестування локалізації.

1.3.3 Порівняльна характеристика підходів до виконання тестування

Для зручності порівняння обох підходів тестування, в ході роботи було систематизовано усі переваги та недоліки кожного з них (табл. 1.1).

Таблиця 1.1 – Порівняльна характеристика ручного та автоматизованого підходів до тестування

Властивість	Ручне тестування	Автоматизоване тестування
Надійність підходу	Можливі помилки в результатах, оскільки можливі помилки через неуважність тестувальника	Практично завжди пройдені тести абсолютно без помилок, оскільки тестувальник не має відношення до процесу виконання тестів
Швидкість виконання тестів	Відбувається набагато повільніше	Оскільки тестування проходить автоматично, результат можна отримати досить швидко
Вартість використання підходу	Для цього методу передбачена лише оплата праці тестувальника	Окрім оплати праці тестувальників, необхідна оплата інструментів для автоматизації
Регулярність виконання тестів	Тести запускаються набагато рідше	Тести можуть запускатися регулярно з визначеними затримками

Отже, автоматизоване тестування однозначно допомагає отримувати результати набагато швидше отримувати кращі та точніші результати. Натомість, ручне тестування дозволяє виконати тестування будь-якого програмного продукту. На мою думку, варто комбінувати обидва підходи для отримання якісного програмного продукту.

1.4 Висновки за розділом

У розділі було розглянуто загальне поняття тестування та його значення у сучасному світі програмних продуктів. Було визначено, що це обов'язковий етап для усіх сфер розробки. Він дозволяє не лише зменшити ймовірність виникнення помилки, яка може нанести шкоду, але й пришвидшити процес роботи із проектом, а також зменшити бюджет, що витрачається при розробці.

Також було розглянуто базові види тестування. Усі вони однозначно необхідні для різних цілей в тестуванні. Також, було детально проаналізовано та порівняно підходи, що використовуються в тестуванні та обрано автоматизований підхід, оскільки він має досить багато переваг.

Для подальшої роботи необхідно виконати аналіз, представлених на ринку, інструментів автоматизованого тестування.

2 АНАЛІЗ ІНСТРУМЕНТІВ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ТА ОБРОБКИ РЕЗУЛЬТАТІВ

2.1 Фреймворки для тестування

Протягом багатьох років Java залишається однією із найкращих мов програмування для розробки та автоматизованого тестування. Із розвитком такого виду тестування, було розроблено безліч середовищ з відкритим вихідним кодом, які побудовані на основі мови програмування Java.

Фреймворки – це програмні продукти, які набагато спрощують створення та підтримку технічно складних проєктів. Основною перевагою таких фреймворків для програмістів є надання стандартизованих та розширюваних засобів для тестування веб-застосунків, або ж для будь-якого іншого програмного продукту. Кожен із таких фреймворків включає в себе величезний набір пакетів, що містять різноманітні бібліотеки, компілятори, інструменти та API, які спрощують роботу з фреймворками в цілому.

Фреймворки, які використовуються в тестуванні здатні забезпечити надійний захист програмних продуктів, так як усі вразливі частини проєкту можна заздалегідь виявити та усунути. В деяких із них також є функції і модулі, що дозволяють зробити процес розробки та тестування набагато швидшими та ефективними [10].

При побудові цього розділу було проведено детальний аналіз найпопулярніших із фреймворків.

2.1.1 JBehave

JBehave (рис. 2.1) — це один із найкращих фреймворків для тестування, а саме BDD, яка ставить на перше місце саме поведінку програми, приділяючи увагу взаємодії з користувачем. В його основі лежить підтримка BDD, що є покращеною версією TDD та ATDD. Цей фреймворк часто використовують новачки для того, щоб краще і швидше зрозуміти BDD. Незважаючи на це, цим

фреймворком часто користуються й справжні професіонали через його легкість, доступність та інформативність.

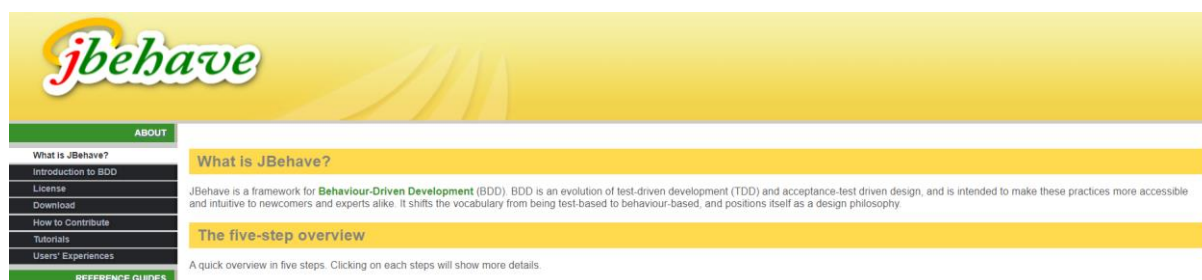


Рисунок 2.1 – Головна сторінка сайту JBehave [11]

Основні дистрибутиви Jbehave:

- JBehave Web (розширення JBehave Core, що надає підтримку доступу або функцій, пов'язаних з мережею);
- JBehave Core (містить усі основні функції для запуску BDD).

Щодо умов та системних вимог до встановлення JBehave, варто зазначити, що найкраще його використовувати в Eclipse IDE, де його версія вище indigo та JDK1.7 і вище. Також для розширення функціоналу можуть знадобитися додаткові файли із розширенням jar, зокрема:

- Commons-lang-2.4.jar;
- Junit-4.1.0.jar;
- Jbehave-core-3.8.jar;
- Freemarket-2.3.9.jar;
- Plexus-utils-1.1.jar;
- Paranamer-2.5.jar;
- Org.apache.commons.collections.jar;
- Org.apache.commons.io.jar.

JBehave має безліч переваг, що обумовлює його популярність, серед подібних фреймворків. Варто зазначити такі його переваги:

- інтуїтивність у використанні фреймворку. JBehave досить зручний та легкий у використанні завдяки тому, що усі деталі інтерфейсу продумані;

- співпраця команд. Як уже раніше було зазначено, JBehave створений для BDD, а отже, в роботі запропоновано інтеграцію між командами програмістів, які працюють над різними програмними продуктами за схожою тематикою, тобто специфікацією;
- напівформальність мови використання. У JBehave наявний словник предметної області, що безумовно допомагає при узгодженні в командах;
- відкритість на усіх етапах. У всіх учасників проекту одна специфікація, що пришвидшує роботу та робить її ефективнішою.

Незважаючи на усі переваги, перелічені раніше, існує один, але надзвичайно значущий недолік – необхідність та важливість спілкування між усіма учасниками створення програмного продукту, у тому числі й керуючими, тестувальниками, розробниками та замовником. Однозначно, якісь недомовленості в команді можуть призвести до помилок у роботі готового програмного продукту.

2.1.2 JUnit

JUnit (рис. 2.2) – це одна з найпопулярніших бібліотек для модульного тестування програмних продуктів, написаних мовою Java. Цей відкритий фреймворк тестування, на відміну від JBehave, зазвичай використовується уже фахівцями, досвідченими розробниками з метою покращення якості коду, написаного мовою програмування Java, а також для пришвидшення роботи програми в цілому, використовуючи при цьому юніт-тести. Основна ціль JUnit у тому, щоб дозволити Java-розробникам писати скрипти, реалізуючи одні і ті ж тести.

The 5th major version of the programmer-friendly testing framework for Java and the JVM

[User Guide](#)[Javadoc](#)[Code & Issues](#)[Q & A](#)[Support JUnit](#)

Рисунок 2.2 – Головна сторінка сайту JUnit [12]

JUnit найефективніший у випадку, коли необхідно покрити невеликий фрагмент коду тестами. Варто зазначити, що JUnit дозволяє за допомогою комбінації з Selenium WebDriver створювати автоматизовані тести сайтів. Окрім того, фреймворк постійно оновлюється в ході написання нового коду та пропонує додавання тестів для покриття усього проекту.

JUnit має ряд особливостей, що значно полегшують роботу тестувальникам:

- набори тестів;
- фікстури (тестові дані);
- класи JUnit;
- Test Runners (інструмент тестування, який вибирає каталог із коду, в який входять модульні тести та їх налаштування і, виконуючи їх, робить запис результату тестів в консоль).

Системні вимоги та умови роботи із JUnit:

- остання версія JDK;
- остання версія JUnit із налаштованим середовищем.

З-поміж переліку інших фреймворків для тестування JUnit виділяється за допомогою таких переваг:

- підтримка інтеграційних та юніт-тестів. Така можливість надається завдяки тому, що JUnit підтримує функції Java 8;

- легкість читання коду. Завдяки зручному інтерфейсу, набагато легше читати код і як наслідок, легше знаходити помилки на ранній стадії, що підвищує надійність вихідного коду;
- сумісність з основними IDE. У більшості із середовищ, наприклад, Eclipse, NetBeans, Maven, Ant та IntelliJIDEA, є можливість сумістити його із фреймворком. Це дає можливість писати та запускати модульне тестування прямо в середовищах програмування;
- сумісність версій. Для тестування можна використовувати тест-кейси, які були написані в старіших версіях фреймворку. Окрім того, оновлена версія JUnit, надає можливість перевірити програму на наявність помилок набагато швидше та ефективніше;
- JUnit для Java. Фреймворк з самого початку був створений як засіб юніт-тестування Java.

Окрім усіх перерахованих переваг у фреймворка JUnit є недолік – його не можна використовувати для тестування залежностей, для цього необхідно використовувати інший інструмент – TestNG.

2.1.3 Serenity (Thucydides)

Serenity (або ж Thucydides) (рис. 2.3) – один з найбільш популярних BDD-фреймворків тестування Java з відкритим вихідним кодом, який дає можливість тестувальникам писати легкі та структуровані автоматизації тестування для проектів.

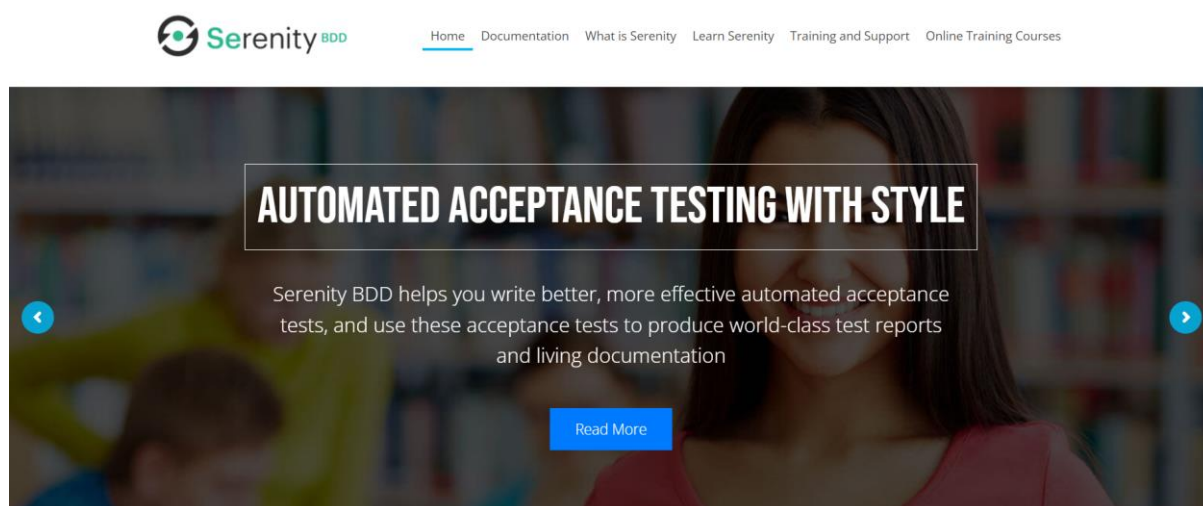


Рисунок 2.3 – Головна сторінка сайту Serenity [13]

Робота з Serenity значно зручніша та ефективніша, зокрема, фреймворк надає можливість створення докладних звітів про тестування та порівнювати результати автоматизованого тестування.

З-поміж інших фреймворків, Serenity виділяється через такі свої особливості:

- Serenity – це Open Source проект;
- автоматичне утворення докладних звітів про результати тестування;
- звіт подається у діловому стилі одразу для кожного тесту.

Потрібно розуміти, що оскільки фреймворк автоматизації повністю заснований на мові програмування Java, то для роботи в Serenity необхідно мати навички в роботі із Java.

Серед системних вимог для роботи Serenity є такі основні пункти:

- версія JDK повинна бути не менша 5;
- необхідно встановити Maven, версія якого повинна бути не старішою, ніж 3.0;
- для простоти експлуатації рекомендується вибирати IDE на зразок Eclipse або Maven.

Як і усі, зазначені раніше фреймворки, Serenity також має свої переваги:

- робота із сервісом REST. Фреймворк Serenity дає змогу тестувальнику створювати тест-кейси для сервісів REST;

- ефективна автоматизація. Фреймворк дає можливість своїми функціями ефективно і швидко працювати, надаючи допомогу у підтримці та створенні власної платформи автоматизації;

- сумісність з іншими фреймворками та інструментами. Serenity досить гнучкий у своєму використанні, більше того, роботу у ньому можна поєднувати з іншими платформами для Java, наприклад, Spring, також можна працювати й з фреймворками такими як Selenium, JBehave та інструментами управління проектами типу JIRA.

За своєю сутністю, Serenity схожий з фреймворком JBehave, тож і недолік у роботі із ним буде таким же – необхідна регулярна комунікація зі всіма учасниками команди для підтримки BDD.

2.1.4 Selenide

Selenide (рис. 2.4) – найпопулярніший open-source фреймворк тестування мовою програмування Java на основі Selenium WebDriver. Selenide – один із основних інструментів для написання адаптивних тест- UI кейсів для Java-додатків, який забезпечував би стабільність роботи. Цей фреймворк значно розширює функціонал WebDriver та JUnit.

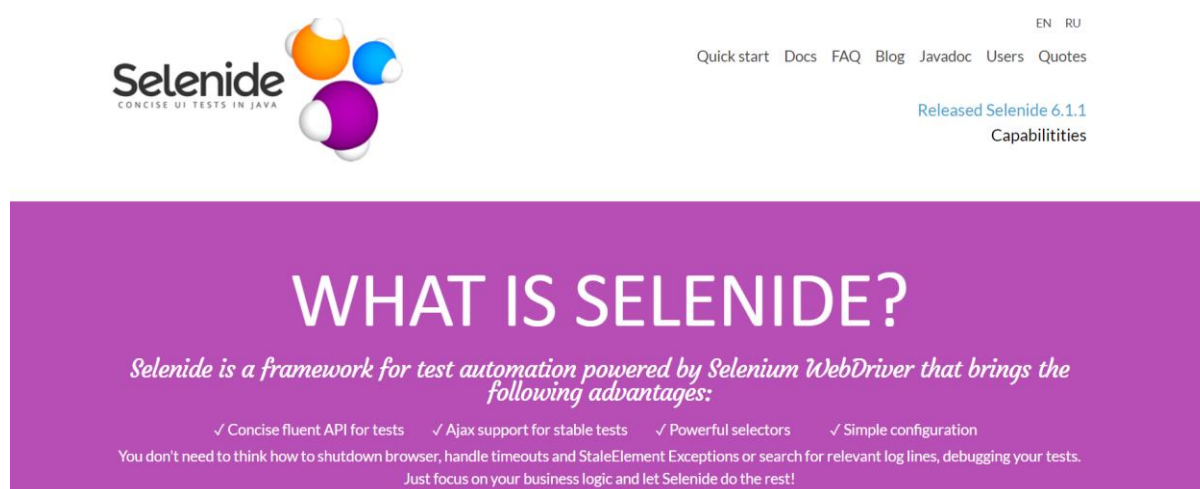


Рисунок 2.4 – Головна сторінка сайту Selenide [14]

Варто детальніше зупинитися на WebDriver. Тож, це один із найвідоміших інструментів для виконання тестування інтерфейсу користувача, наприклад, тестування веб-технологій Ajax, який може допомогти вирішити помилки з затримками. Фреймворк Selenide надзвичайно легкий для ознайомлення та роботи із ним.

Серед особливостей Selenide можна зазначити такі:

- фреймворк дозволяє приступити до роботи одразу після установки;
- не потрібно писати багато коду для автоматизації;
- економія часу та ресурсів.

Для установки даного фреймворку знадобиться:

- версія JDK не нижче 5;
- версія Maven 3.0 або вище;
- IDE (варто зазначити, що для цього ідеальним може бути Eclipse,

оскільки Eclipse та Maven добре сумісні у роботі).

Selenide однозначно має безліч переваг, окрім тих, що були зазначені раніше, зокрема:

- ефективна робота з Ajax та затримками. За допомогою цього, тести, отримані на виході будуть більш точними та стабільними, а отже, й ефективними;
- рідкісні випадки затримки. Це відбувається завдяки тому, що не потрібно писати довжелезний код;
- сумісність з AngularJS. Фреймворк здійснює підтримку тестів, які були створені за допомогою AngularJS;
- економія часу. Так як практично, Selenide – це дочірній фреймворк від Selenium, то більш команд там скорочено.

2.1.5 TestNG

TestNG (рис. 2.5) – це один із найпопулярніших і найпотужніших фреймворків Java-тестування, з-поміж тих, що використовуються з метою

створення інтеграційних, функціональних та юніт-тестів. На даний момент, TestNG є надзвичайно вагомим конкурентом для різних фреймворків, зокрема й для JUnit, оскільки при роботі з ним доступні різноманітні анотації та функції, які роблять процес тестування значно ефективнішим та швидшим.



Рисунок 2.5 – Головна сторінка сайту TestNG [15]

Фреймворк TestNG хороший інструмент в усіх типах тестування програмних продуктів, зокрема, його використовують для модульного, інтеграційного, функціонального тестування та інших. Так само як і JUnit, TestNG є фреймворком, який використовується для написання тестів мовою програмування Java, це дає можливість використовувати додаткові інструменти, які допомагають досліджувати отримані результати тестування.

TestNG має безліч особливостей, які надзвичайно допомагають в процесі тестування, зокрема:

- фреймворк надає можливість тестування залежностей;
- конфігурація тестів в TestNG є досить гнучкою;
- TestNG надає надійну модель виконання тестів;
- TestNG сумісний із BeanShell, що робить цей фреймворк ще більш гнучким у використанні.

Фреймворк TestNG надає підтримку тестувальникам за допомогою окремих функцій Java, зокрема, угруповання, послідовності та параметризації. Така особливість фреймворку дає можливість робити тестування більш

адаптивним, при цьому практично перемагаючи фрейворки, що використовувалися раніше.

Умови встановлення TestNG:

- остання версія JDK;
- остання версія IDE, наприклад, Eclipse.

Безумовно, підсумовуючи раніше сказане, можна дійти висновку, що у фреймворці TestNG є власні переваги, такі як:

- паралельне тестування. Фреймворк розроблений таким чином, що він дозволяє можливість створення та запуску тестування одразу декількох фрагментів коду;
- створення звітів. TestNG надає тестувальникам можливість після проходження тесту створювати власний звіт у форматі HTML;
- можливість групування тестів. Фреймворк надає можливість групувати тести, а також впорядковувати їх, що значно покращує логіку тестування програмного продукту;
- пріоритетність тестів. За допомогою вже згаданої функції параметризації, а також використання анотації, можна легко встановлювати пріоритети у проходженні тестування.

Незважаючи на досить великий список переваг фреймворку TestNG, існує досить незначний недолік – на встановлення цього фреймворку необхідно витрати чимало часу.

2.1.6 Gauge

Спеціально для тестування із використанням мови програмування Java, а саме для проведення автоматизованого тестування вебзастосунків, було створено і ще один фреймворк. Gauge (рис. 2.6) – це безкоштовний фреймворк, з відкритим кодом, який дає можливість виконання автоматизації тестування Java, яке, як уже зазначалося раніше, значно пришвидшує процес тестування, а

також підвищує його якість. Також, варто зазначити, що цей фреймворк один із найпопулярніших для тестування веб-сайтів для мобільних пристроїв.

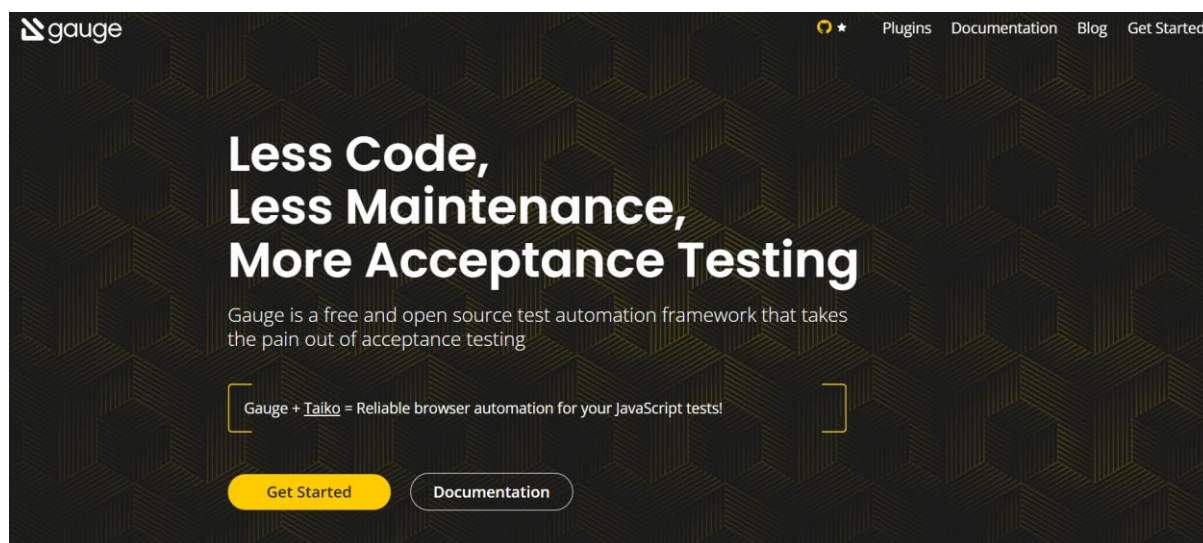


Рисунок 2.6 – Головна сторінка сайту Gauge [16]

Фреймворк пропонує підтримку одразу декількох мов програмування, зокрема Ruby, JavaScript, Java, C# та Python.

В роботі із Gauge є декілька особливостей, які варто зазначити:

- фреймворк має дуже простий і гнучкий синтаксис, що робить його використання легшим;
- у своїй роботі фреймворк використовує ділову мову;
- надзвичайно важливим є те, що Gauge здійснює підтримку одразу декількох мов програмування та різноманітних платформ;
- на сайті фреймворку можна знайти багато різноманітних плагінів, що явно полегшать процес тестування.

Для того, аби встановити фреймворк Gauge необхідно:

- встановити останню версію Maven;
- серед системних вимог зазначено, що ПЗ Microsoft Windows повинно бути не нижче 10 версії;
- для нормальної роботи фреймворк вимагає мінімум 512 MB оперативної пам'яті.

Варто зауважити, що якщо потрібно запустити тест через командний рядок, або ж термінал, необхідно виконати налаштування плагіна фреймворку Gauge з IDE.

Фреймворк однозначно має безліч переваг, деякі з них є унікальними, зокрема:

- співпраця Agile командами. Фреймворк Gauge надає безліч переваг командам протягом розробки DevOps. Серед них варто відзначити можливість зворотного зв'язку;
- перевага бізнес-логіки. Завдяки такій властивості, читабельність скриптів стає набагато кращою;
- гнучкість фреймворку. Ця властивість забезпечується тим, що відбувається підтримка безлічі мов програмування;
- сумісність з різними IDE. Фреймворк дозволяє створювати тести у будь-якому середовищі програмування;
- створення звітів. Система дозволяє створювати паралельні тести та звіти;
- робота з великими об'ємами даних. Фреймворк дозволяє виконувати тестування, зберігаючи при цьому його якість та точність, у роботі із великими об'ємами даних.

У своїй роботі, не зважаючи на значні переваги, фреймворк Gauge має також недоліки:

- відносна непопулярність фреймворку. Незважаючи на те, що Gauge має дуже багато переваг, проте тестувальники досить рідко його використовують через те, що фреймворк був створений відносно нещодавно;
 - відсутність підтримки паралельної реалізації специфікацій в IntelliJ.
- Це відбувається через те, що Gauge не може бути розділений скриптами.

2.1.7 Порівняльна характеристика фреймворків для тестування

В ході роботи було проведено порівняльний аналіз даних фреймворків (табл. 2.1).

Серед усіх можливих плюсів різних фреймворків можна виділити 3 основних, які максимально покращать роботу з кодом:

- паралелізація – головний критерій відбору, адже так можна прискорювати виконання наших тестів, в деякі моменти це може бути досить критично, як після глобального оновлення роботи сайту;
- параметризація – допомагає використовувати всі можливості фреймворку, економити час на написання важких тестів та робити їх менш ресурсоємними;
- життєвий цикл – допомагає налаштовувати тести в залежності від його життєвого циклу, тобто анотоційна складова.

Таблиця 2.1 – Порівняльна характеристика різних фреймворків для тестування

Властивість Фреймворк	Паралелізація (паралельний запуск тестів)	Життєвий цикл	Параметризація тестів
JBehave	Присутня	Є BDD (розробка проходить по іншому)	Відсутня
JUnit	Присутня	Великий вибір анотацій	Присутня
Serenity	Присутня	Є BDD (розробка проходить по іншому)	Відсутня
Selenide	Присутня	Великий вибір анотацій	Відсутня
TestNG	Присутня	Великий вибір анотацій	Присутня
Gauge	Присутня (але з проблемами)	Є більш орієнтованим на читабельність кода	Відсутня

Таким чином можна виділити лідерів, таких як TestNG, Junit, Selenide. Але за рахунок того, що TestNG має перевагу в налаштуванні при паралелізації, через наявність власного виконуваного файлу, легкого описання процесу, більш продуманої праметризації, ніж Junit, та більшою кількістю анотацій з-поміж своїй конкурентів – він і стає лідером в даному порівнянні.

2.2 Інструменти автоматизації для комп'ютерів

Для того, аби розробити якісний програмний продукт, необхідно виконувати різні типи тестування, зокрема, навантажувальне, функціональне, системне та інші. На сьогоднішній день значну роль в виконанні такої роботи становить автоматизація. Для цього використовується величезна кількість різноманітних інструментів для створення автоматизованих тестів, що дозволяє компанії-розробнику досягати максимальної якості в результаті роботи над власним програмним продуктом [17]. У цьому підрозділі буде наведено аналіз різних, найпопулярніших інструментів автоматизованого тестування.

2.2.1 Selenium

Selenium (рис. 2.7) – це один з найпопулярніших, найвідоміших та найефективніших інструментів для виконання автоматизації тестування сайтів та вебзастосунків. Незважаючи на те, що цей інструмент був створений дуже давно, він досі користується надзвичайною популярністю, оскільки Selenium постійно розвивається, спрощуючи при цьому механіку написання скриптів.

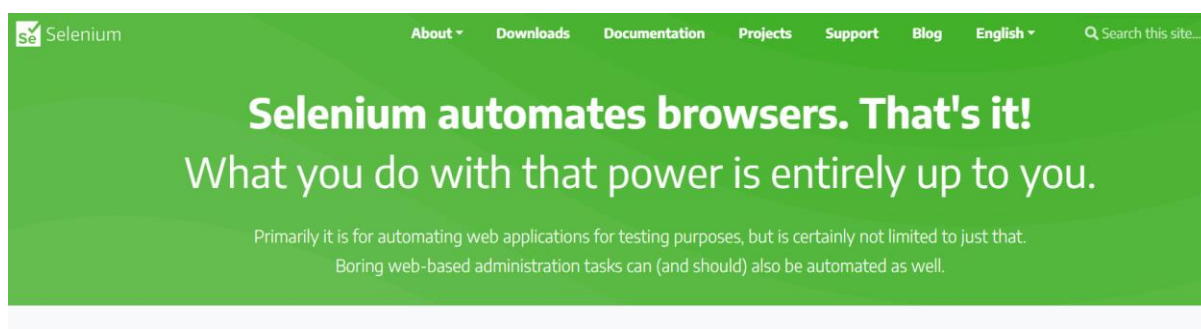


Рисунок 2.7 – Головна сторінка сайту Selenium [18]

Саме від Selenium пішли деякі інші інструменти, зокрема, Robot Framework, Watir, Protractor, Katalon Studio. Важливо, що даний інструмент автоматизації тестування дозволяє використовувати різноманітні мови

програмування, у тому числі, Java, Python, C#, PHP, а також ця платформа дає можливість проводити автоматизацію тестування на базі різних операційних систем (Mac, Linux, Windows) з використанням найпопулярніших браузерів, таких як Firefox, IE, Chrome, тощо.

З того, що було зазначено раніше, можна зробити висновок, що платформа має безліч переваг, зокрема, її гнучкість, а також інструмент надає можливість для написання складних скриптів для автоматизації тестування. Незважаючи на це, в Selenium є досить суперечливий недолік – для ефективної роботи із цим інструментом необхідно мати досить великий багаж знань для того, аби написати бібліотеки та фреймворки, які будуть забезпечувати виконання функцій у процесі автоматизованого тестування програмного продукту.

Для автоматизації роботи в веб-браузері у роботі було використано спеціальний драйвер – Selenium WebDriver (рис. 2.8). Тобто, фактично, це бібліотека, яка дозволяє розробляти програми, які можуть керувати роботою браузерів. Selenium WebDriver відноситься до набору інструментів, які широко використовуються в сфері тестування, особливо, коли необхідно виконати тестування у різних браузерах. Необхідно зауважити, що Selenium WebDriver не можна використовувати для автоматизації звичайних програм, цей драйвер використовується лише у браузерах. Selenium WebDriver є одним із найпопулярніших та найпотужніших інструментів для автоматизованого тестування, оскільки має досить широкі можливості.

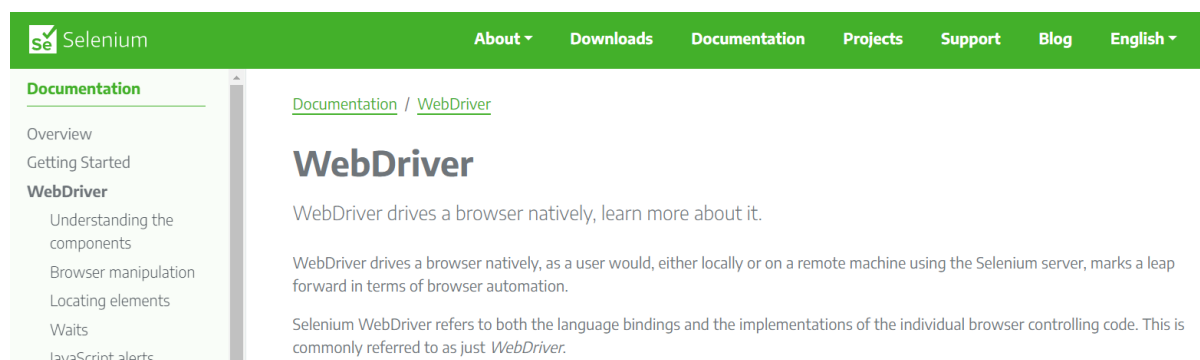


Рисунок 2.8 – Selenium WebDriver [19]

Цей драйвер має декілька основних переваг, зокрема:

- дає можливість тестування у різних найпопулярніших браузерах, наприклад, до Chrome, Safari, Opera, Firefox, IE тощо;
- підтримує різні мови програмування, зокрема, Java, C#, PHP, Ruby, Perl, Python, що дозволяє тестувальникові самостійно вибирати мову, яка є зручною для нього;
- легке середовище для розробки – існує безліч функцій, що значно пришвидшують роботу із цим драйвером;
- із цим драйвером можливе паралельне тестування одразу в різних браузерах в один і той же час.

2.2.2 Unified Functional Testing

Unified Functional Testing (що в перекладі означає – комплексне функціональне рішення для тестування) (рис. 2.9) – це теж один із найпопулярніших інструментів для автоматизації тестування, проте, на відміну від попереднього розглянутого Selenium, ця платформа розроблена на комерційній основі, тобто використання його усіх функцій - платне. UFT пропонує для автоматизації тестування спеціальний набір функцій, призначених для роботи із веб-сервісами, сайтами, API, графічним інтерфейсом вебзастосунків, із використанням надзвичайно великої кількості платформ. В арсеналі інструменту є також розширені можливості, які пропонують функції розпізнавання об'єктів на зображеннях, багаторазові компоненти тестування, а також створення звітності автоматизованого тестування.

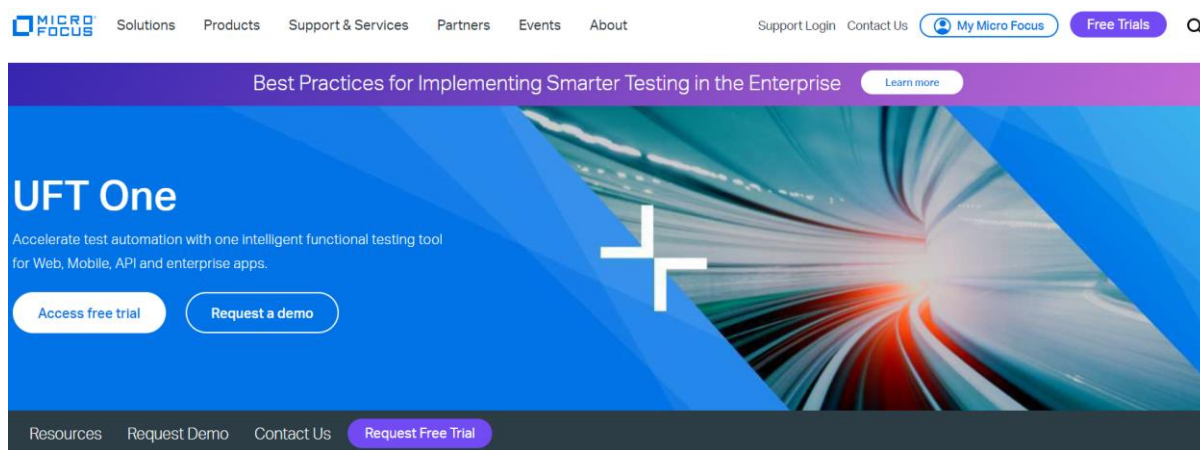


Рисунок 2.9 – Головна сторінка сайту Unified Functional Testing [20]

В роботі з інструментом, можливе збереження інформації про виконане тестування, а також керувати необхідними об'єктами, оскільки для його роботи використовується Visual Basic Scripting Edition.

2.2.3 Katalon Studio

Katalon Studio (рис. 2.10) – це і ще один надзвичайно корисний інструмент для автоматизації тестування як сайтів і веб-сервісів, так і мобільних програм. Цей інструмент має схожість із уже згаданим Selenium, а також із Appium, аналіз якого буде поданий в наступному підрозділі. Зокрема, це проявляється у тому, що Katalon Studio має можливість інтегрованої автоматизації тестування програмних продуктів.

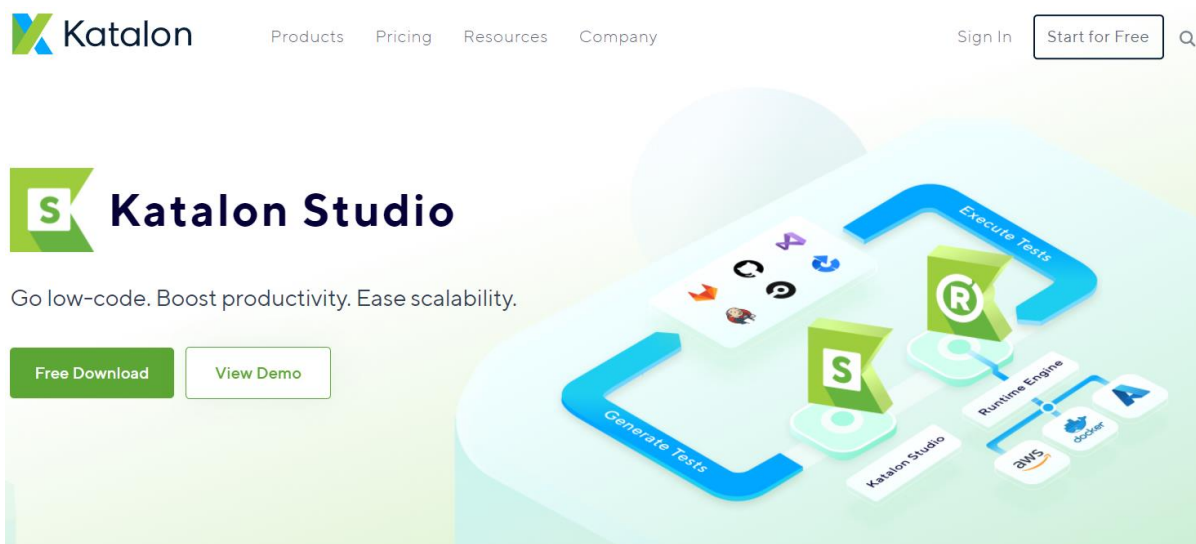


Рисунок 2.10 – Головна сторінка сайту Katalon Studio [21]

Проте, на відміну від роботи із Selenium, для роботи з Katalon Studio достатньо базових знань процесу тестування. Хоча, скоріше за все, розробити автоматизоване тестування програмного продукту в цьому інструменті може розробити навіть новачок. Це можливе завдяки наявності функції Object Spy. В будь-якому випадку, інструмент задовольняє потреби усіх тестувальників, через наявність уже готових скриптів, або ж бібліотек.

Katalon Studio надає повну звітність, як опис виконаного тестування. Такі звіти забезпечуються за допомогою вбудованої функції Katalon Analytics включають різноманітні графіки, діаграми та метрики. Окрім того, інструмент інтегрується в CI/CD, а отже, під час автоматизації процесу тестування можлива інтеграція і з іншими інструментами, зокрема, JIRA, Jenkins, qTest, Git.

2.2.4 IBM Rational Functional Tester

IBM Rational Functional Tester (рис. 2.11) – це інструмент, який призначений для того, аби ефективно виконувати функціональні та регресійні типи тестів. Цей інструмент підтримує тестування різними мовами програмування, зокрема, Java.

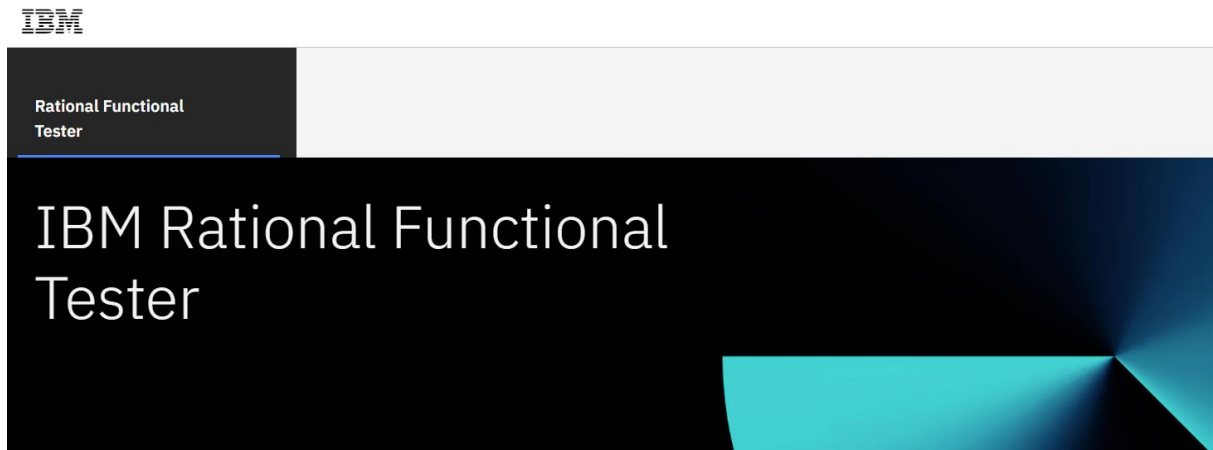


Рисунок 2.11 – Головна сторінка сайту IBM Rational Functional Tester [22]

Інструмент IBM RFT має унікальну функцію, яка дозволяє виконувати запис процесу автоматичного тестування, а також надає можливість візуалізувати усі ці дії, зокрема, у вигляді скріншотів. Така функція має назву Storyboard Testing.

2.2.5 TestComplete

TestComplete (рис. 2.12) – це ще один із найпопулярніших засобів для автоматизації процесу тестування різних вебзастосунків. Цей інструмент підтримує роботу за допомогою таких засобів, як JavaScript, Python, C++ Script, тощо. TestComplete дещо схожий із Katalon Studio, оскільки, цей інструмент також дозволяє використання ключових слів, а також проводити тестування, кероване даними. Не мало важливою перевагою TestComplete як інструменту для процесу автоматизації тестування є надання можливості запису усього процесу з можливістю повторювання.

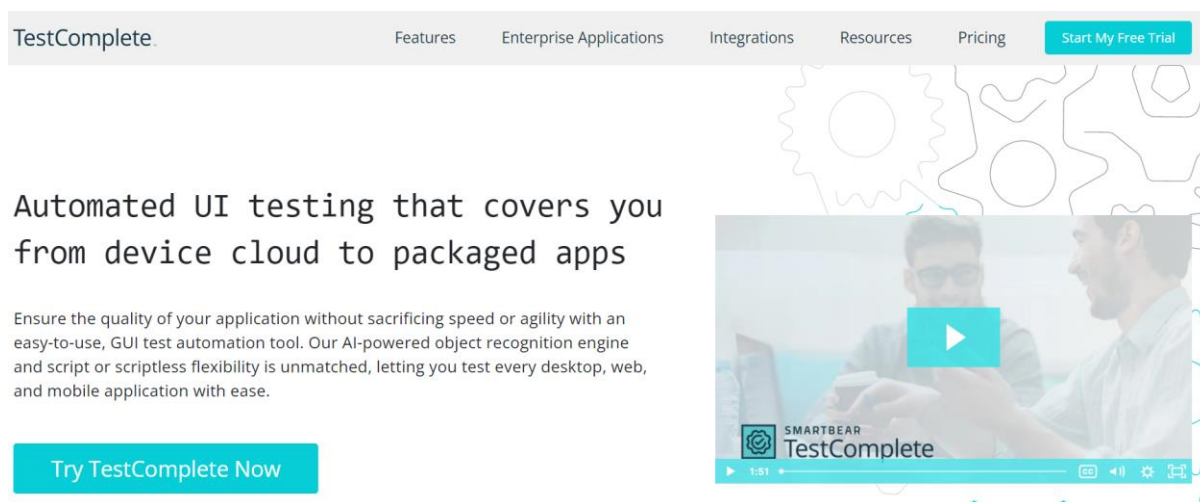


Рисунок 2.12 – Головна сторінка сайту TestComplete [23]

Також, TestComplete має схожі функції з UTF, зокрема, можливість розпізнавання об'єктів GUI, в процесі виконання якої відбувається процес виявлення, а як результат відбувається оновлення інтерфейсу користувача.

2.2.6 Порівняльна характеристика інструментів автоматизації для комп'ютерів

Як результат проведеного аналізу усіх переваг та недоліків, була проведена їх систематизація (табл. 2.2).

Основними критеріями відбору будуть наступні пункти:

- ОС – це дуже важливий фактор, оскільки, користувачі можуть використовувати різні ОС та в залежності від цього робота застосунків може змінюватись;
- підтримувані мови програмування – важливий при виборі мови програмування для розроблюваного проєкту;
- можливість неперервної інтеграції разом з Jenkins – найважливіший критерій, який допомагає робити тестування та обробляти результати;
- вартість.

Таблиця 2.2 – Порівняльна характеристика інструментів автоматизації для комп'ютерів

Властивість Інструмент	Операційна система	Мови програмування, які підтримує інструмент	Можливість використання неперервної інтеграції	Тип ліцензії та вартість
Selenium	Windows, Linux, OS X	Всі	Присутня	Безкоштовна
UFT	Windows	VBScript	Присутня	Платна
Katalon Studio	Windows, Linux, OS X	Java/Groovy	Присутня	Безкоштовна
IBM RFT	Windows, Linux	Java	Присутня	Безкоштовна
TestComplete	Windows	JS, python, VBScript, JScript	Присутня	Безкоштовна

Selenium – очевидний переможець через те, що він є безкоштовним, підтримує всі можливі ОС та є дуже великою платформою, яка має велику кількість інформації, яка допоможе та спростить розробку у майбутньому.

2.3 Інструменти автоматизації для мобільних пристроїв

У сучасному світі, де практично в кожній людині в будь-якому віці уже є мобільний пристрій, виникає потреба у автоматизації процесу тестування й на них. У цьому випадку автоматизація дозволяє замінити трудомістке ручне тестування на набагато зручніший та швидший спосіб. Окрім того, автоматизоване тестування дає змогу виконувати різні, недоступні для ручного, типи тестування, зокрема, регресійне, що є дуже важливим для мобільних пристроїв.

У цьому підрозділі буде описано декілька найвідоміших та найефективніших інструментів для автоматизації процесу тестування.

2.3.1 Appium

Зрозуміло, що в сучасному світі важко уявити людину, в якій б не було мобільного телефону, відповідно вимоги до тестування таких пристроїв теж дуже високі. Проте, потрібно зазначити, що зараз існує дуже широкий вибір різноманітних мобільних телефонів, які відрізняються операційною системою, розширенням екрану, графічним інтерфейсом тощо. Тому для правильного і ефективного тестування необхідно використовувати автоматизовані тести для застосунків на смартфонах.

Існує безліч інструментів для таких задач, один із них – Appium (рис. 2.13). Він підходить для автоматизації як застосунків на смартфонах із операційною системою Android, так і IOS. Це абсолютно безкоштовний інструмент, який отримує HTTP-запити у форматі JSON від клієнтів і тоді переписує їх в залежності від платформи, на якій працює. Така ж форма роботи відбувається і при використанні Selenium WebDriver.



Рисунок 2.13 – Головна сторінка сайту Appium [24]

Для коректної роботи Appium необхідно, щоб виконувалися деякі системні вимоги, зокрема:

- для IOS:

- Mac OSX;
- XCode w/ Command Line Tools.
- для Android:
- Mac OSX or Windows or Linux;
- Android SDK ≥ 16 .

Варто зауважити, що в Appium є і ще й інші переваги, зокрема:

- Для того, щоб реалізувати процес автоматизації тестування, не потрібно виконувати будь-яких додаткових змін в програмному коді розроблюваного програмного продукту, або перекомпілювати її.
- Через те, що Appium є універсальним інструментом, можливе використання різних мов програмування та фреймворків.

2.3.2 Detox

Detox – це ще один надзвичайно популярний інструмент для автоматизованого тестування за допомогою JavaScript. Одна із головних переваг Detox – це те, що через те, що застосунки на iOS та Android максимально схожі, оскільки генеруються з одного й того ж коду, а отже, й тести можна використовувати одні і ті ж відразу для двох платформ. Також, цей інструмент надзвичайно зручний для програм, які були створені за допомогою React Native. Detox дає можливість роботи як з емуляторами, так і з фізичними пристроями.

Принцип тестування, який називається «сірий ящик» - це те, що відрізняє цей інструмент від всіх інших. В даному випадку Detox, отримуючи доступ до внутрішніх механізмів програмного продукту, може порівнювати поведінку на верхньому (зовнішньому) та нижньому рівнях застосунку. Detox може навіть звертатися до пам'яті для того, щоб відстежувати виконувані процеси тестування.

Принцип сірого ящика однозначно допомагає боротися з нестійкістю, яка може давати знати про себе при наскрізному тестуванні.

Цікаво, що такий принцип ще й може пришвидшити процес проходження тестування, оскільки на відміну від «чорного ящика», він здатен уникати створенню різноманітних пауз.

Detox має можливість використовувати нативні драйвери прямо на пристроях, підтримуючи роботу за допомогою JSON. Це допомагає виключити потребу у WebDriver.

2.3.3 Ranorex

Ranorex (рис. 2.14) – це, на відміну від усіх попередніх інструментів, платний комплексний інструмент для тестування вебзастосунків різних типів. Ranorex підходить як для новачків, так і для професіоналів, оскільки інструмент дозволяє виконувати автоматизацію процесу тестування як за допомогою програмування, так і взагалі без використання скриптів. Цей інструмент для процесу автоматизації тестування дає можливість проводити його як в емуляторах, так і на справжніх пристроях.

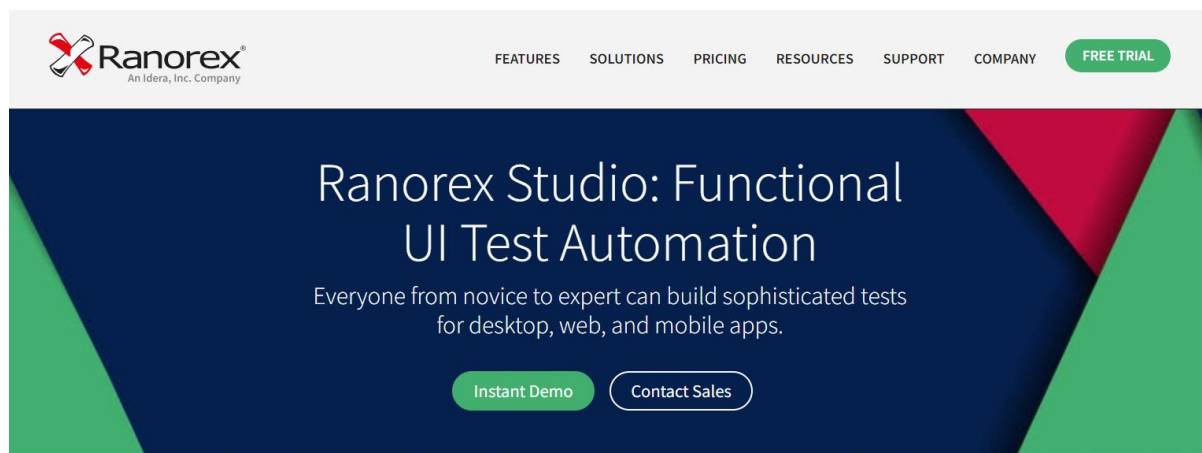


Рисунок 2.14 – Головна сторінка сайту Ranorex [25]

Ranorex надає доступ до власного центру управління, де надається можливість створення тестів, їх налаштування та запуску одразу на різних

пристроях та в різних середовищах. Окрім того, інструмент дає можливість паралельного тестування на будь-яких девайсах.

Також при використанні цього інструменту можлива інтеграція з Git для контролю версій, а також з CI (Jira). Дозволяється також завантаження даних із таких сервісів як SQL, CSV, Excel. Це реалізовано за допомогою наявного data-driven тестування.

На відміну від Detox, Ranorex підтримує три підходи до тестування (білого, сірого та чорного ящика).

2.3.4 Selendroid

Selendroid (рис. 2.15) – це інструмент автоматизації тестування, який використовується для Android застосунків.

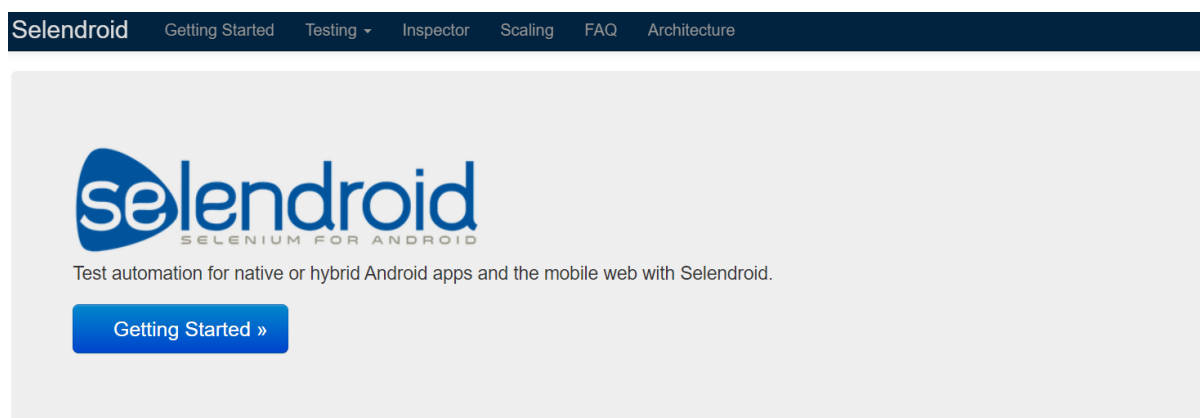


Рисунок 2.15 – Головна сторінка сайту Selendroid [26]

Selendroid має надзвичайно велику кількість переваг, хоч і призначений лише для Android, зокрема:

- сумісність із JSON;
- автоматизація процесу тестування програмного продукту, не вимагає змін в ньому;
- легкий та зручний інтерфейс користувача, який допомагає знайти усі необхідні елементи за допомогою різних типів локаторів;

- підтримується паралельне тестування декількох пристроїв.

2.3.5 Порівняльна характеристика інструментів автоматизації для мобільних телефонів

Для того, аби краще зрозуміти відмінності між інструментами автоматизації, їх основні переваги були приведені до таблиці (табл. 2.3).

Основними критеріями для відбору є:

- операційні системи – виконання тестування на всіх можливих операційних системах є важливим аспектом для інструментів;
- мови програмування – важливий при виборі мови програмування для розроблюваного проєкту;
- паралелізація – оскільки кількість мобільних пристроїв та версій занадто велика, тому тут обов’язково повинна бути присутня паралелізація для швидкого виконання тестування.

Таблиця 2.3 – Порівняльна характеристика інструментів автоматизації для мобільних телефонів

Властивість Інструмент	Операційна система	Мови програмування, які підтримує фреймворк	Паралелізація (паралельний запуск тестів)
Appium	Android, iOS, WebView	Всі	Дозволяє паралелізацію
Detox	Android, iOS, WebView	JavaScript	Лише 1 застосунок
Ranorex	Android, iOS, WebView	C#	Дозволяє паралелізацію
Selendroid	Android, WebView	Java	Лише 1 застосунок

Таким чином Appium та Ranorex є лідерами, але, оскільки, Ranorex є платною платформою, то нашим лідером стає Appium.

2.4 Системи звітності тестування

Як результат роботи над будь-яким етапом розробки програмного продукту, представляються звіти. Тестування не стало виключення, адже яким би ефективним не було тестування, саме запис процесу в деталях, а саме, запис усіх тестових випадків, дає зрозуміти, яка робота була виконана і не дозволяє втратити жодну із її деталей. Окрім того, це буде надзвичайно корисним у випадку роботи в команді.

Selenium не підтримує створення звітів, хоча для цього у вільному доступі є безліч додаткових інструментів. але це досить легко зробити за допомогою відповідних інструментів.

2.4.1 Автоматичні системи звітування

Як уже зазначалося, є безліч методів для створення звітів про тестування за допомогою різноманітних інструментів. Серед таких:

- TestNG. На цій платформі доступна бібліотека, яка дозволяє створювати звіти автоматично, зберігаючи їх у спеціальній папці. Ця функція є повністю автоматичною. Для цього можна використовувати спеціальні інтерфейси, зокрема, `ITestListener` та `IReporter`;

- також, доступний файл, який як правило називається `index.html`, в якому міститься звіт, в якому зібрана абсолютно вся інформація про помилки, час виконання тестування, усі журнали звітів;

- окрім того, є файл, який як правило називається `Emailable-report.html`, в якому можна побачити підсумкову оцінку окремих тестів;

- JUnit. На цій платформі звіти генеруються за допомогою HTML. Для цього процесу використовується спеціальний клас `TestWatcher`, який у свою чергу викликає методи про проходження, або не проходження тесту, порівнюючи очікувану та явну поведінку програмного продукту;

- також є додаткові анотації `@BeforeClass` і `@AfterClass`, які дають можливості відкриття та закриття звіту;

- бібліотека `Extent Report`. Цей засіб, так само як і попередній, створює звіти HTML, при цьому дещо розширюючи доступні функції, які полегшують роботу із звітами, особливо тим, що надаються діаграми, графіки та інші графічні елементи. Окрім того, надаються функції запису історії тестування, фільтрації, знімків екрану та інші;

- Allure. Цей фреймворк має у своєму арсеналі адаптер, який зберігає тестову інформацію у XML-файлах, а потім перетворює їх у звіт HTML. Він також дає можливість створювати графіки.

2.4.2 Звітування, що виконується вручну

Такий спосіб створення звітування про тестування, насправді дуже хороший альтернативний спосіб, оскільки можна самостійно записувати усе, що необхідно. Хоча насправді такий метод може вимагати багато часу та зусиль. Для цього можна використовувати PDF-файл із Java API IText.

Для того, аби зручно оформити запис результатів тестування вручну є декілька порад:

- перш за все необхідно вказати, чи вдалим був тест;
- варто робити скріншоти екрану з результатами виконання тестування;
- необхідно вказати причину виникнення помилки;
- іноді варто вказати список підключених методів та класів;
- іноді досить корисно надавати відсоткову оцінку результатів виконання тесту;
- обов'язково необхідно вказати інформацію про середовище та інструменти тестування;
- за можливості варто надавати графічний опис тестування.

Проте, незважаючи на те, що фактично при використанні цього методу є можливість надати максимально детальну інформацію, але все ж варто пам'ятати, що оформлення звіті тестування може займати занадто багато часу, що робить це менш ефективним.

2.5 Висновки до розділу

В розділі розглянуто різноманітні фреймворки, інструменти та засоби звітності для процесу автоматизації тестування. У ході написання цієї частини роботи було обрано системи, які будуть використовуватися для написання власного програмного продукту, зокрема, було обрано фреймворки та інструменти, з опорою на власний аналіз запропонованих на ринку.

Для того, аби розробити власний програмний продукт, який буде задовільняти потреби тестувальників, в подальшому необхідно обрати додаткові інструменти розробки.

3 МЕТОДОЛОГІЇ ТА ТЕХНОЛОГІЇ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ПРОДУКТУ

3.1 Мова програмування

Java — одна із найпотужніших мов програмування в усьому світі. Ця мова використовується для різноманітних цілей, зокрема, для написання різноманітних програмних продуктів для персональних комп'ютерів, мобільних застосунків, обробки великої кількості даних, різноманітних систем тощо. Також, Java використовується й в тестуванні. Варто зазначити, що за даними TIOBE index [27] за листопад 2021 року, Java займає третє місце (рис. 3.1) у загальному рейтингу мов програмування за кількістю написаних рядків на ній. Окрім того, за даними компанії Oracle [28], яка володіє Java, дану мову програмування використовують більше, ніж 12 мільйонів розробників зі всього світу, що робить Java однією з найпопулярніших мов програмування.

Nov 2021	Nov 2020	Change	Programming Language		Ratings	Change
1	2	▲		Python	11.77%	-0.35%
2	1	▼		C	10.72%	-5.49%
3	3			Java	10.72%	-0.96%
4	4			C++	8.28%	+0.69%
5	5			C#	6.06%	+1.39%

Рисунок 3.1 – Рейтинг найпопулярніших мов програмування за TIOBE index за листопад 2021 року [27]

Щодо автоматизації тестування, то потрібно зазначити, що вибір мови програмування не є одним із вирішальних факторів. Для цього можна використовувати будь-яку мову, зокрема, Java, Python, C# тощо.

Тож, при виборі мови програмування для тестування, вирішальними стали такі переваги Java:

- широкий вибір різноманітних IDE. Надзвичайно важливим є середовище для програмування. Для мови Java існує величезний вибір таких, зокрема, IDE від IntelliJ, NetBeans тощо. Важливо, щоб IDE допомагала у створенні автоматичних тестів, виконуючи частину роботи самостійно. Особливо це відноситься до виправлення синтаксису. Тут на допомогу приходять функції автодоповнення коду;

- використання PageFactory. PageFactory, як клас бібліотеки Selenium, ініціалізує необхідні об'єкти сторінки, звертаючись до них лише тоді, коли виникає потреба в цьому, тобто тоді, коли це зазначено в коді. На практиці ж, при роботі без PageFactory, може з'явитися помилка у WebDriver, так як відбуватиметься ініціалізація абсолютно усіх елементів, але при цьому частина з них може бути не знайдена на сторінці. Тобто при використанні PageFactory, за умови, що в коді немає згадки про той чи інший об'єкт сторінки, його не буде ініціалізовано, хоч він і відноситься до класу сторінки. PageFactory Java значно спрощує код для автоматизації на Selenium, дозволяючи писати прості для розуміння тести;

- надзвичайна популярність мови програмування. Як уже раніше було зазначено, Java – одна із найпопулярніших мов програмування. На практиці ж це означає, що при виникненні будь-якої незрозумілої ситуації, завжди можна знайти допомогу як в своєму оточенні, так і в мережі Інтернет, зокрема на сайті Stack Overflow.

Проте, варто зазначити, що під час виконання роботи, було виділено також ряд недоліків мови програмування Java, серед яких:

- простота написання й читання коду. Програмний код, написаний мовою Java, набагато важче читати і розуміти, ніж той же код написаний, наприклад, на мові Python, адже у Python набагато легший синтаксис, зокрема, зрозуміти такий код можна навіть із базовими знаннями англійської. Окрім того, документація до Java досить важко написана і далеко не завжди є зрозумілою та корисною. Проте, як уже раніше було зазначено, через велику

популярність цієї мови, відповідь на складні запитання практично завжди можна знайти на спеціалізованих сайтах в Інтернеті;

- вказівник `null pointer`. При роботі із Java інколи виводиться `stack trace`, тобто список методів, які були використані до моменту, коли виникла помилка, і це далеко не завжди дає чітке розуміння причини виникнення помилки, а інколи такий вивід взагалі стає абсолютно непотрібним. Хоча, парто зазначити, що середовища для програмування, зокрема й IntelliJ часто намагається допомогти, але не завжди такі повідомлення дають повну картину для розуміння помилок;

- використання лише для автоматизації тестування. Java – безумовно один із лідируючих інструментів для автоматизації тестування. Проте, варто розуміти, що насправді спеціалізація цієї мови програмування досить вузька у тестуванні, тож навряд чи її можна ефективно використовувати для Ви ніколи не використовуватимете Java в інших областях тестування, наприклад, для CI, тобто для інтеграції різних частин коду між собою, для тестування продуктивності, безпеки тощо. Java дозволить автоматизувати лише тестування графічного інтерфейсу необхідної програми.

3.2 Фреймворк тестування

TestNG - один з найпопулярніших фреймворків тестування Java. Він має безліч переваг над конкурентами, такі як:

- велика лінійка зрозумілих і корисних анотацій: `BeforeSuite`, `AfterSuite`; `BeforeTest`, `AfterTest`; `BeforeClass`, `AfterClass`; `BeforeMethod`, `AfterMethod`; `BeforeGroups`, `AfterGroups` ; `Test`. Завдяки цій ієрархії можна легко будувати передумови, постумови, різні ініціалізації та гарний код.

- також є велика кількість корисних параметрів, виділю кілька з особистого досвіду:

- `dependsOnMethods` - Допомагає керувати тестами на основі їх успішності;

- `timeOut` - корисний параметр, якщо є ліміти часу виконання чи нам потрібно перевіряти час виконання;
- `description` - додається для формування гарного та зрозумілого звіту;
- `enabled` - за допомогою цієї анотації не треба прибирати тест з роботи чи коментувати його, якщо нам потім потрібно буде повернутися до роботи з ним;
- `alwaysRun` - дуже важливий параметр, який найчастіше буде використовуватися в After-анотаціях, для завершення якоїсь дії, або якщо існує обов'язковий тест у тестовому наборі.
- параметризоване тестування:
- завдяки анотації `DataProvider` можна параметризувати тести, а саме робити їх унікальними для деякого об'єкта, що тестується, таким чином економити час на написання тестів для інших випадків. А головним плюсом є економія місця і легка читабельність коду;
- також слід виділити анотацію `Factory`, яка дозволяє запустити набір тестів з різними наборами даних. Фактично вона використовує `DataProvider`, але не для окремого тесту, тому в деяких випадках, коли нам потрібно від початку до кінця обробити якийсь набір тестів, з певним набором даних, ця анотація буде дуже корисна, тому що вона буде економити час та місце.
- спосіб запуску тестів:
- дуже гарна система роботи з запуском тестів. Окремий XML-файл, в якому можна описати різні набори для тестування, для різного виду тестування чи місця, що тестується;
- також в цьому файлі є можливість керувати порядком запуску тестових класів та методів;
- додавати спеціальні параметри до наших тестів, які будуть використовуватися для певних налаштувань.
- запуск паралельних тестів. Також є можливість налаштовувати паралелізацію різних класів чи методів, за допомогою чого можна прискорювати роботу наших тестів;

- звіт. Також TestNG формує звіт, згідно результатів тестування. Виділяє успішні, пропущені, поламани тести, час їх виконання та мінімальний звіт об помилках.

Аналізуючи всі ці плюси, можна сміливо казати, що TestNG є чудовим вибором для роботи з автоматизованим тестуванням. Його привілеї говорять самі за себе, того для роботи було обрано саме його.

3.3 Сервер, який дозволяє тестам використовувати екземпляри веббраузера, запущені на віддалених машинах

Іноді виникає потреба запуску тестів в декількох потоках, чи на різних браузерах, чи різних операційних системах.

Selenium Grid - це кластер, який включає в себе декілька Selenium-серверів, на котрих можна організувати роботу для запуску відразу декількох паралельних тестів на різних браузерах та операційних системах.

Основний принцип його роботи це те, що необхідний центральний сервер - hub, на якому можливо створювати вузли - node. Тому в самій програмі відбувається з'єднання з самим сервером, передаєм туди дані, та вже сам сервер розподіляє по вузлам тести, в залежності від умови браузера чи операційної системи.

Так як, основною метою роботи є вдосконалення процесу тестування, то не зайвим буде викристання Selenium Grid.

За його допомогою можна легко запускати тести одразу в декількох потоках на різних браузерах.

Тому в наступних випадках цей інструмент буде дуже корисний:

- запуск тестів на різних браузерах;
- економія часу та запуск на різних нодах;
- запуск тестів на різних операційних системах.

Та також великим плюсом є те, що кластер дуже легко встановлюється та взаємодіє з іншими інструментами програми.

3.4 Звітність

Після виконання тестування, потрібно обробити результат. Тобто дослідити, чи всі тести були успішно закінчені. Якщо ні, то не погано було б дізнатися де та в яких місцях.

Для змістовного звіту було обрано Allure. Цей інструмент дозволяє робити змістовні звіти.

Звісно для того, щоб тримати гарний звіт, треба буде інтегрувати інструмент до програми та робити деякі анотації. Але результат того вартий. Цей інструмент значно полегшує роботу з обробкою та аналізом результатів.

За допомогою звітності можна позначати проблемні тести, ввести історію успішності, переглядати статистику тестів.

Великим плюсом є те, що можна виділяти кожен окремий крок, який тільки захочемо, та валідувати його успішність. Разом з тим і передавати ці дані до звіту та не витрачати час на їх пошук у програмі.

І ще один з плюсів це те, що можна додавати скріншоти будь якого місця, в будь який час. Також можна додавати файл з текстом, наприклад результатами пошуків, логами, API запитів тощо.

Також є можливість налаштувати сортування тестів для нашого звіту, незалежно від класу до якого вони відносяться, що є останнім важливим плюсом цього інструменту.

3.5 Хмарна платформа для веб- та мобільного тестування

Для тестування веб-сайтів та мобільних застосунків в браузерях, у роботі використовується хмарна платформа – BrowserStack. Ця система надає широкий спектр різноманітних операційних систем та браузерів, а також величезний вибір (рис. 3.2) емуляторів мобільних пристроїв як на операційній системі Android, так і на IOS, а також їх стандартних браузерів[29].

List of Real Mobile & Tablet Devices

Android		iOS	
Samsung		Google	
Galaxy S21	11.0	Pixel 5	12.0
Galaxy S21 Ultra	11.0	Pixel 5	11.0
Galaxy S21+	11.0	Pixel 4	11.0
Galaxy S20	10.0	Pixel 4	10.0
Galaxy S20+	10.0	Pixel 4 XL	10.0
Galaxy S20 Ultra	10.0	Pixel 3	10.0
Galaxy A51	10.0	Pixel 3a XL	9.0
Galaxy A11	10.0	Pixel 3a	9.0
Galaxy Note 20	10.0	Pixel 3 XL	9.0
Galaxy Note 20 Ultra	10.0	Pixel 3	9.0
Galaxy S10	9.0	Pixel 2	9.0
Galaxy S10+	9.0	Pixel 2	8.0
Galaxy S10e	9.0	Pixel	8.0
Galaxy S9+	9.0	Pixel XL	7.1
Galaxy S8+	9.0	Pixel	7.1
Galaxy A10	9.0	Nexus 6	6.0
Galaxy Note 10	9.0	Nexus 6	5.0
Others		Others	
		OnePlus	
		9	11.0
		8	10.0
		7T	10.0
		7	9.0
		6T	9.0
		Xiaomi	
		Redmi Note 9	10.0
		Redmi Note 8	9.0
		Redmi Note 7	9.0
		Motorola	
		Moto G9 Play	10.0
		Moto G7 Play	9.0
		Moto X 2nd Gen	6.0
		Moto G 2nd Gen	5.0
		Razr	4.0

Рисунок 3.2 – Список емуляторів та мобільних пристроїв, доступних на BrowserStack [29]

В цій системі є безліч можливостей, і деякі з них роблять її унікальною серед інших, зокрема:

- система дозволяє переглянути власний сайт на різноманітних пристроях, але, варто зауважити, що така функція можлива лише на стандартних браузерях для телефонів;
- існує можливість виконати так званий масовий тест, коли будуть отримані скріншоти, для цього можна навіть виставити спеціальний таймер, який буде відраховувати час для отримання скріншотів;
- за додаткову плату також можна отримати доступ на будь-якій з операційних систем до будь-якого із браузерів;
- дуже важливою у тестуванні є можливість отримати доступ до API, що дозволяє реалізовувати тести в бажаних браузерах на бажаних операційних системах.

В залежності від ваших потреб ви можете використати любі мобільні девайси, різні види операційних систем та браузері.

В залежності від рівня вашої підписки ви можете робити тести одразу на різних девайсах, для економії часу.

Також в BrowserStack присутній перегляд результатів тестування. Зручний звіт та відеозапис всього тесту робить аналіз легким і доступним. Дуже об'ємна бібліотека з інформацією, щодо використання цього інструменту робить його гарним рішенням для даної роботи.

3.6 Інструмент автоматизації для запуску скриптів і тестування мобільних вебзастосунків

Для більшості продуктів мобільне тестування є першорядним, тому що в майже в кожній людині є власний смартфон, завдяки якому вона може завжди заходити до інтернету та відвідувати сайти.

Таким чином мобільне тестування є дуже важливим аспектом для деяких продуктів і в цьому нам допоможе такий інструмент для мобільної автоматизації, як Appium.

Основні плюси:

- вона є мультиплатформлена, а саме незалежно від операційної системи будь то iOS чи Android, нам не потрібно буде використовувати окремий інструмент для кожної окремої платформи;
- інструмент використовує версію інструмента WebDriver, тому всі команди будуть ідентичні до Selenium, який в свою чергу використовує цей само інтерфейс;
- дуже легко переробити тести з комп'ютерного тестування в мобільне так, як вони використовують ідентичні інтерфейси;
- кросплатформність, що означає, що нам не треба буде писати для кожної платформи окремий тест, а він буде автоматично підлаштовуватись під певну платформу.

Таким чином, ми, використовуючи WebDriver, можемо легко перероблювати наші тести під мобільні платформи. А мультиплатформеність надасть нам різних варіантів для роботи з різними типами операційних систем. І найголовніше це те, що маючи написаний тест для комп'ютерного застосування, можна з легкістю його переробити під все види платформ за допомогою кроссплатформності.

3.7 Система автоматичного збирання

Програма містить в собі багато доповнень, тому потрібно використовувати системи автоматичного збирання.

В цьому питанні мій вибір впав на Gradle через наступні переваги:

- Gradle набагато швидше ніж Maven, тому коли буде використовуватися безперервна інтеграція, він пришвидшить час складання проекту;
- зручна інтеграція модулів;
- можна створювати додаткові скрипти/завдання.

3.8 Інструмент для автоматизації роботи в веб-браузері

Для взаємодії коду з браузером треба використовувати спеціальний інструмент, який буде виступати в ролі посередника.

Таким інструментом є Selenium WebDriver.

Його переваги:

- продукт є безкоштовним, на відміну від своїх прямих конкурентів;
- він є унікальним, тому його можна легко використовувати з різними видами фреймворків та інструментами тестування;
- кроссплатформеність, незалежно від операційної системи наші тести будуть працювати;

- кроссбраузерність, всі існуючі браузері можуть бути автоматизовані за допомогою цього інструменту;
- легко інтегрувати разом з Selenium Grid;
- легкий в освоєнні.

Таким чином можна зробити висновки, що цей інструмент є найкращим для наших цілей і в даному наборі інструментів він буде гарно співпрацювати з усіма іншими елементами.

3.9 Інструмент для безперервної інтеграції

Часто виникає проблема в необхідності перевірки продукту чи окремо функціоналу кожен певний час.

Для цього використовується інструменти для безперервної інтеграції, за допомогою котрих можна запускати певні тести, за певним розкладом.

Таким інструментом є Jenkins. За його допомогою можна легко встановлювати безперервні інтеграції для автоматизованого тестування, будь то комп'ютерні тести чи мобільні, встановлювати ланцюги автоматизації, налаштувати різні причини для запуску окремої інтеграції

Його головні плюси:

- безкоштовний;
- легко налаштування система безперервної інтеграції;
- система розширень, плагінів;
- легко використовувати програмне забезпечення для відстеження змін (Git, GitLab) для використання коду.

Jenkins є дуже великим продуктом, який надає максимально корисні функції для роботи безперервної інтеграції, обробки результатів та валідації.

3.10 Середовище реалізації розроблюваного програмного продукту

IntelliJ IDEA - це комерційне IDE для програмування на різних мовах, створена компанією JetBrains. Відповідно до даних отриманих із ресурсу PYPL [30], IntelliJ IDEA входить в десятку найпопулярніших середовищ розробки в листопаді 2021 року (рис. 3.3). Індекс популярності різноманітних середовищ програмування розраховувався за допомогою підрахунку кількості запитів для завантаження IDE у системі Google. Чим більше пошукових запитів, тим більш популярне IDE.

Rank	Change	IDE	Share	Trend
1		Visual Studio	29.24 %	+3.5 %
2		Eclipse	13.91 %	-2.9 %
3	↑	Visual Studio Code	12.07 %	+3.3 %
4	↓	Android Studio	9.13 %	-2.5 %
5		pyCharm	8.43 %	+0.7 %
6		IntelliJ	6.7 %	+0.8 %
7		NetBeans	4.82 %	-0.3 %
8	↑↑	Sublime Text	3.49 %	-0.2 %
9	↓	Xcode	3.37 %	-1.2 %
10	↓	Atom	3.25 %	-0.5 %

Рисунок 3.3 – Рейтинг найпопулярніших середовищ програмування за даними PYPL за листопад 2021 року [30]

Зрозуміло, що при розробці будь-якої системи, у тому числі автоматизації тестування, розробнику важливо, щоб IDE було зручним, а також мало

повноцінний доступ до усіх бібліотек та функцій. IntelliJ IDEA має безліч переваг, зокрема:

- зручність у використанні. У цьому IDE окрім звичних підсвітки синтаксису та декількох буферів обміну, є і безліч інших корисних функцій, що значно полегшують та прискорюють процес розробки. Серед таких варто зазначити, автоматичне додавання необхідних імпортів, їх оптимізація (у тому числі з видаленням), автозаповнення із урахуванням необхідних типів тощо;
- інтеграція із Git (розподіленою системою керування версіями файлів та спільної роботи [31]);
- постійне оновлення версій та можливість установки нових плагінів;
- ергономічний дизайн самого середовища програмування.

IntelliJ IDEA — це в першу чергу IDE для Java, хоча насправді вона розуміє та навіть надає допомогу при написанні коду на таких мовах як SQL, JPQL, HTML, JavaScript та багатьох інших, а також дозволяє редагувати код, написаний не за допомогою Java, всередині рядкових літералів Java коду.

Враховуючи переваги IntelliJ IDEA, саме це середовище було обрано для власного проекту, а саме його версію .

3.10.1 Системні вимоги для реалізації програмного продукту за умови використання IntelliJ IDEA

Зрозуміло, що для роботи на базі OS Windows, а також для забезпечення коректності виконання програмного коду IntelliJ IDEA має свої системні вимоги для роботи [32]:

- Microsoft Windows 8/10 (64-bit version);
- Мінімум 2 GB RAM , рекомендовано для пришвидшення обробки – 8 GB RAM;
- Мінімум 2.5 GB пам'яті на диску, на який встановлюється середовище, рекомендовано – SSD;
- Мінімальне розширення екрану – 1024 x 786.

3.11 Висновки до розділу

В цьому розділі було зазначено додаткові інструменти для розробки власного програмного продукту, а також вимоги до роботи з ними.

Тепер, коли було вказано усю методологічну основу роботи, необхідно переходити до безпосередньої розробки програмного продукту.

4 ОРГАНІЗАЦІЙНА ЧАСТИНА

В цьому розділі наведені технічні та якісні характеристики розробленого програмного продукту.

4.1 Загальні положення

В процесі досліджень було розроблено фреймворк для тестування мобільних та комп'ютерних застосунків, а саме веб-застосунків. Фреймворк повинен слугувати, як початок для розробки автоматизованого тестування нового продукту, як для мобільною так і для комп'ютерної версії, без додаткового покращення чи додавання нових інструментів.

Розроблений продукт має в собі наступні переваги:

- універсальність;
- розширюваність;
- придатність до модернізації та масштабування;
- модульність та структуру легку для сприйняття;
- готовність для роботи з моменту написання першого тесту.

4.2 Основний процес, який повинен виконувати програмний продукт

Основним завданням продукту є спрощення написання автоматизованих тестів для різних платформ, пристроїв та браузерів.

Програма повинна забезпечити користувачеві наступні привілеї:

- легка ініціалізація WebDriver;
- можливість обирати між мобільним тестуванням та комп'ютерним;
- обробку помилок;
- можливості до логування процесу тестування;
- можливість аналізу звіту після виконання програми;

- можливість паралельного запуску тестування.

4.3 Загальна модель роботи системи та ролі користувачів системи

Нижче буде наведена загальна модель роботи та взаємодії користувача з комп'ютером.

В межах реалізації роботи фреймворку для автоматизованого тестування можна виділити 2 основних розділи:

- ініціалізація WebDriver;
- тестування.

Всі тести запускаються за допомогою XML, тому описані тести послідовно запускаються з файлу. Вже потім відбувається ініціалізація Selenium WebDriver, в залежності від параметрів з XML або середовища (локального чи на сервері).

В наступній частині будуть реалізовані тести, які вже були раніше написані, використовуючи PageObject - класи, які реалізуються за допомогою Selenium WebDriver (рис. 4.1).

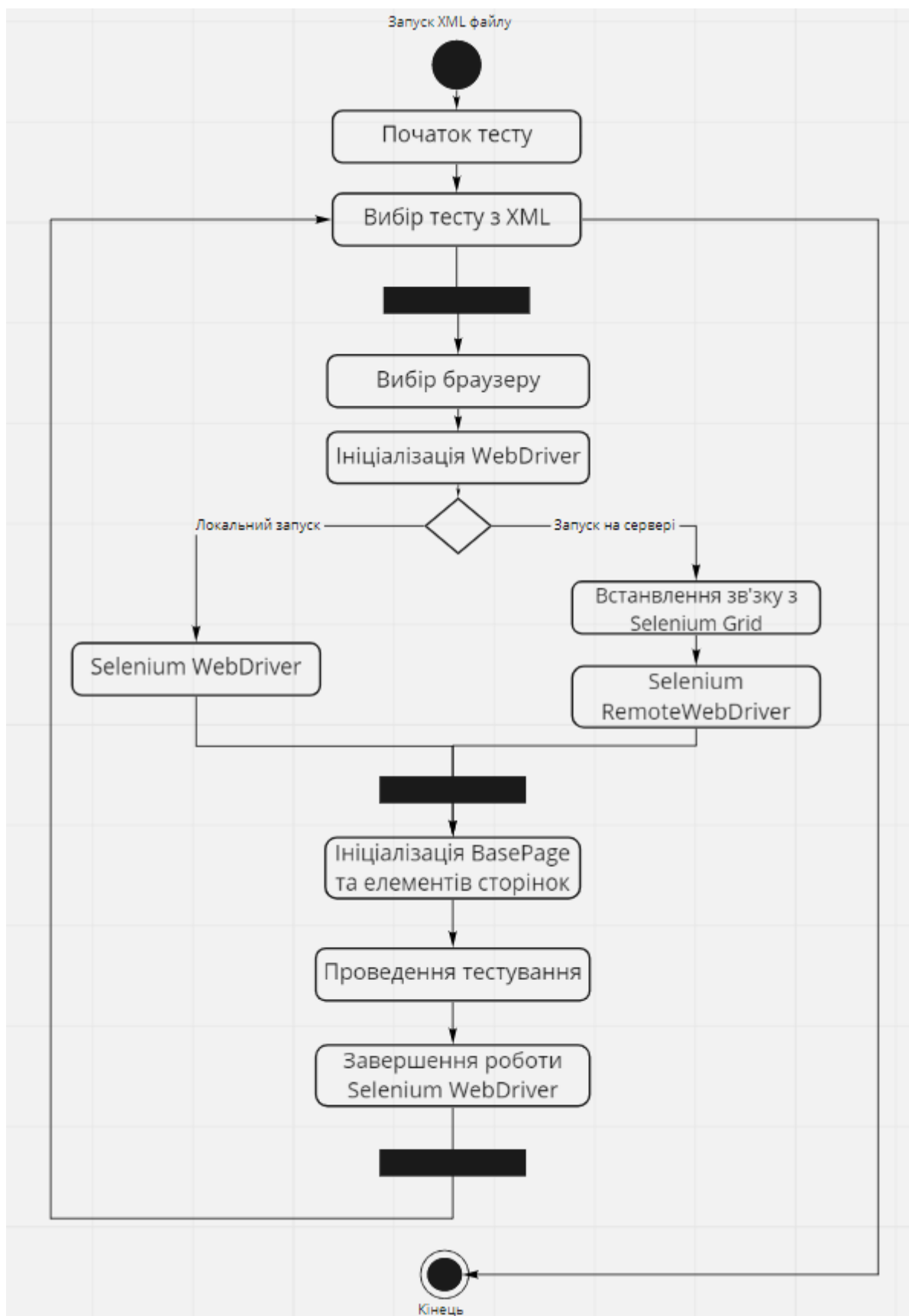


Рисунок 4.1 – Робота Selenium WebDriver

Нижче буде наведена загальна модель роботи та взаємодії користувача з мобільним пристроєм.

В межах реалізації роботи фреймворку для мобільного автоматизованого тестування можна виділити 2 основних розділи:

- ініціалізація RemoteWebDriver;
- тестування.

Всі тести запускаються за допомогою XML, тому описані тести послідовно запускаються з файлу. Обов'язково треба додатково описувати моделі телефону. Вже потім відбувається ініціалізація RemoteWebDriver, в залежності від параметрів з XML ініціалізується телефон.

В наступній частині будуть реалізовані тести, які вже були раніше написані, використовуючи PageObject - класи, які реалізуються за допомогою RemoteWebDriver (рис. 4.2).

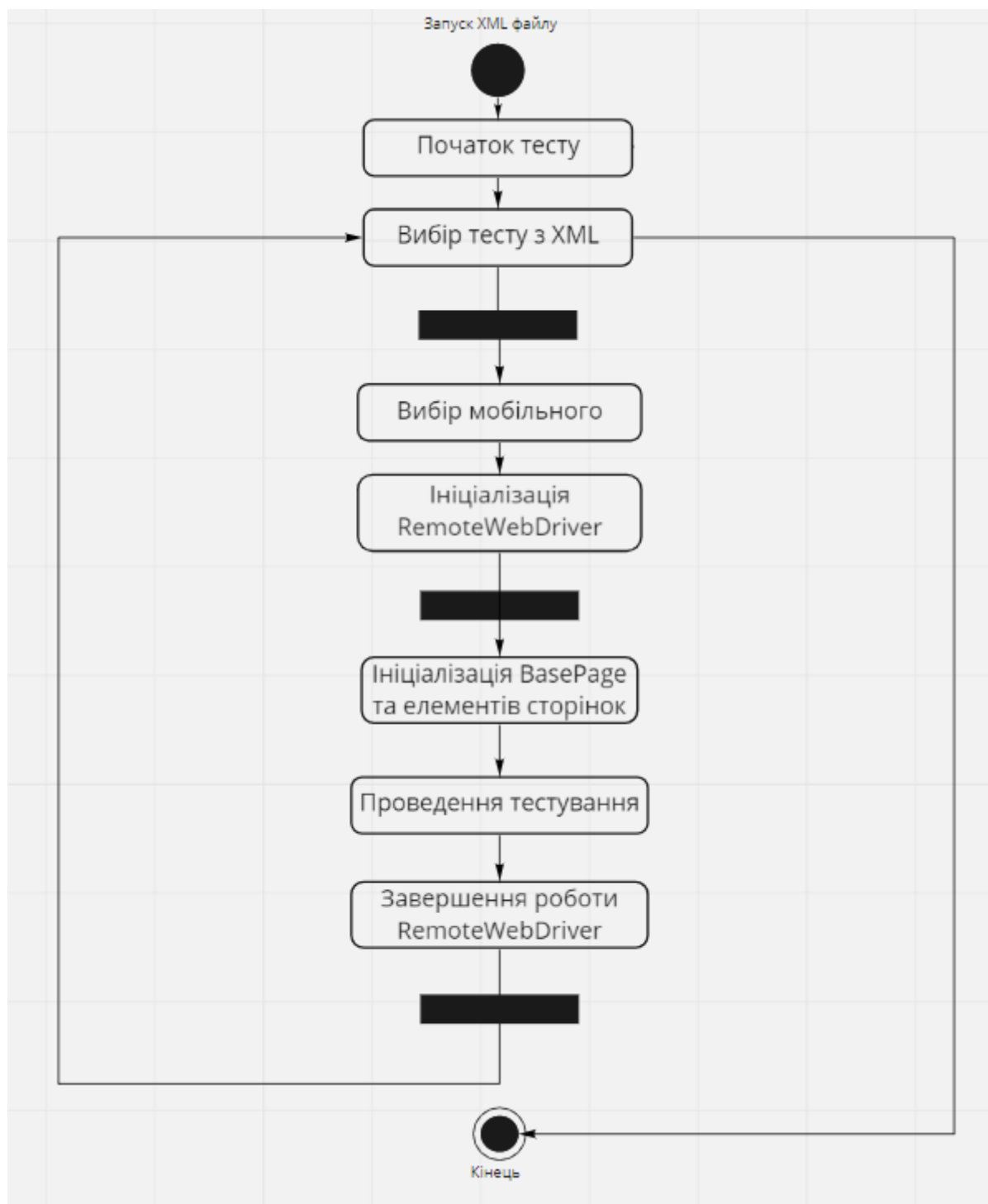


Рисунок 4.2 – Робота RemoteWebDriver

4.4 Взаємодія користувача з фреймворком

Ключовою складовою ПП є процес написання PageObject класів та безпосередньо самих тестів, адже це і є ТЗ розроблюваного продукту.

Передумова: користувач встановив всі необхідні компоненти та завантажив додаткові бібліотеки все за допомогою Gradle.

Після цього користувач має змогу приступити до реалізації своїх цілей:

- на комп'ютері (рис. 4.3);



Рисунок 4.3 – Схема написання тестів для комп'ютера

- на мобільному пристрої (рис. 4.4).



Рисунок 4.4 – Схема написання тестів для телефону

5 РЕАЛІЗАЦІЯ СИСТЕМИ

5.1 Основні класи

5.1.1 Клас Base

Клас Base - це допоміжний клас. Він використовується виключно для того, щоб була можливість додавати знімки екрану та текстові документи до системи репортингу, а сама Allure.

5.1.2 Клас BaseTest

Клас BaseTest - є класом-батьком для всіх тестових класів.

В ньому виконується:

- ініціалізація браузера, в залежності від параметра, який передається при запуску певного тесту;
- встановлення зв'язку з Selenium Grid, якщо того потребує сама програма, в залежності де запускається програма. В протилежному випадку програма запускає WebDriver, в залежності від обраного браузера;
- після закінчення тестування, завершує роботу драйвера задля успішного наступного запуску.

В цьому класі також додаються спеціальні “слухачі” для змоги робити скріншоти та виконувати логування.

Також в цьому класі задається с посилання на Selenium Grid.

5.1.3 Клас BasePage

Клас BaseTest - є класом-батьком для всіх майбутніх класів Page Object.

В ньому виконується ініціалізація класу DriverHelper, за допомогою котрого обробляються всі елементи с Page Object класу, та деякі інші функції. Вона виконується в декілька типів, в залежності від часу, потрібного на очікування.

5.1.4 Клас BaseBlock

Клас BaseBlock- є класом-батьком для всіх класів, які є частиною сторінки. Це унікальна функція Selenium 3, яка дозволяє роботу з окремими блоками всередині певної сторінки, без прив'язки до неї

В ньому виконується ініціалізація класу DriverHelper, за допомогою котрого обробляються всі елементи с Page Object класу, та деякі інші функції. Вона виконується в декілька типів, в залежності від часу, потрібного на очікування.

А також тут можна Прописувати стандартні методи для цього блоку, так як він унаслідкується від HtmlElement класу, який виступає елементом і містить багато цікавих і корисних функцій.

5.1.5 Клас BaseMobile

Клас BaseMobile – це допоміжний клас. Він використовується виключно для того, щоб була можливість підключитися до BrowserStack, використовуючи наші дані, які будуть зазначені в browserstack.properties.

Тут перебувають методи для читання з файлу browserstack.properties.

5.1.6 Клас BaseBrowserStack

Клас BaseBrowserStack – це допоміжний клас для майбутніх тестових класів.

Він, в залежності від отриманих даних, налаштовує браузерні можливості та встановлює телефон разом з операційною системою, та після цього підключається до серверу і надає доступ для тестування.

Також для коректного завершення роботи тут стоїть закриття драйверу.

5.1.7 Клас BasePage

Клас BaseTest - є класом-батьком для всіх майбутніх класів Page Object.

В ньому виконується ініціалізація класу DriverHelper, за допомогою котрого обробляються всі елементи с Page Object класу, та деякі інші функції. Вона виконується в декілька типів, в залежності від часу, потрібного на очікування, але так як драйвер ініціалізується по іншому, а саме за допомогою BrowserStack, то виникає необхідність у цьому класі.

5.1.8 Клас DriverHelper

Це допоміжний клас, який буде використовуватися для роботи з елементами та іншими функціями та методами WebDriver.

Цей клас містить:

- ініціалізатори, які в залежності від вхідних параметрів будуть змінювати час затримки;
- допоміжні функції, які потрібні для того, щоб тесті не падали при кожній помилці, чи затримці завантаження сторінки. Тобто кореляція коректної роботи програми за допомогою WebDriverWait, Try-Catch та логування, для обробки помилок.

5.2 Допоміжні класи

5.2.1 Класи для API

Клас BaseAPI - є класом для роботи з API. Він містить деякі функції для початку побудови запитів.

Для створення запитів, потрібно створити клас та успадкуватися від BaseAPI. Далі треба встановити деталі голови та тіла запиту.

Клас ParseData - є класом для обробки результатів запитів. Він містить всередині допоміжні методи для отримання даних з відповіді запиту.

5.2.2 Клас даних

Так як вставляти прямі дані в методи не є гарною практикою виникла необхідність розробки методу для отримання даних з XML файлу. Адже всі дані зберігаються в одному місці і це є правильно.

Класи `UserData` та `DataManager` - в своєму об'єднанні встановлюють зв'язок з файлом `UserData.xml` та беруть потрібні дані за ключовими полями. Таким чином всі дані зберігаються в одному місці і робимо програму більш універсальною

В залежності від специфіки продукту можна змінювати дані чи поля чи файли для роботи з ними.

5.2.3 Класи-слухачі

`ScreenshotListener` - клас, який допомагає нам дізнаватися о результатах тестування. Він створений для того, щоб на різні стани нашого тесту, а саме:

- успіх;
- провал;
- пропущено;
- провал за часом.

Програма зробила знімок екрану та додавала його до Allure звіту.

`TestLogs` - клас, який допомагає нам дізнаватися о станах тестування. Він створений для того, щоб на різні стани нашого тесту, а саме:

- початок;
- успіх;
- пропущено;
- провал ;
- провал за часом;
- початок нового тесту;

- кінець тесту.

Програма зробила логування тим самим покращила наше прийняття та пошук помилок для валідації програми.

5.3 Клас для логування

Log - клас, який використовує log4j за для того, щоб була можливість зробити процес роботи нашої програми більш детальним.

Це потрібно для того щоб, краще розуміти в чому саме проблема виникає при роботі програми.

Також існує файл log4.properties, в якому писанні правила виводу логування.

5.3.1 TestNG XML

Так як в процесі розробки було використано фреймворк тестування такий, як TestNG, то і тести запускаються за допомогою спеціального файлу XML, в якому ми можемо описувати різноманітні варіації та набори тестів.

Існує 2 типи XML файлів:

- комп'ютерні;
- мобільні.

ВИСНОВКИ

У сучасному світі, який заповнили різноманітні пристроїв, з'явилася гостра необхідність у розробці різноманітних застосунків, вебсервісів сайтів, тощо. Для того, аби зробити цикл розробки максимально ефективним, необхідно не забувати про етап тестування. Правильно організований та реалізований цей етап може допомогти зробити реалізацію програмного продукту набагато швидшою, більш економічною та якіснішою.

В ході виконання роботи, було досліджено різні підходи до тестування – ручний та автоматизований, а також було виконано порівняльний аналіз в ході якого було порівняно переваги й недоліки кожного з них. За результатами роботи було визначено, що при розробці програмного продукту найкраще використовувати змішаний підхід до тестування, оскільки тільки в такому випадку надається можливість покриття усього проекту різноманітними методами тестування.

Перед безпосередньою розробкою програмного продукту було виконано порівняння усіх найпопулярніших інструментів та методів для процесу автоматизації тестування програмних продуктів, а також визначено найкращі з них для використання у власному проекті.

Для дослідження було створено декілька тестових зразків, які працювали на різних платформах та за допомогою різних інструментів.

При розробці програмного продукту для автоматизації процесу тестування було створено технічне завдання, а також пройдено етап розробки. В створеного фреймворку є значні переваги, зокрема:

- має властивість легкої доповнюваності через структуру проекту, та можливості page factory та page object;
- використання передуючих технологій дозволяє розширяти можливості фреймворку та додавати нові засоби чи методи;
- при правильному оформленні, фреймворк надає чудовий звіт, а логування допоможе в пошуках помилки;

- проект можна легко завантажити та використовувати за допомогою безперервного виконання;
- проект містить спеціальні методи для роботи с WebDriver, які значно покращують роботу та аналіз помилок;
- проект можна використовувати як для мобільних так і для веб тестувань.

Розроблений програмний продукт доступний за посиланням:
https://github.com/ytymofieiev/Test_Framework

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Kaner C. Lessons Learned in Software Testing: A Context-Driven Approach 1st Edition / C. Kaner, J. Bach, B. Pettichord., 2011. – 600 p.
2. Black R. Advanced Software Testing / Rex Black., 2014. – 536 p. – (2).
3. Axelrod A. Complete Guide to Test Automation : Techniques, Practices, and Patterns for Building and Maintaining Effective Software Projects / Arnon Axelrod., 2018. – 529 p.
4. Beizer B. Black-Box Testing: Techniques for Functional Testing of Software and Systems / Boris Beizer., 1995. – 320 p.
5. Software Test Engineering with IBM Rational Functional Tester: The Definitive Resource / [eds.: C. Davis, D. Gouveia, D. Chirillo]., 2009. – 696 p.
6. García B. Mastering Software Testing with JUnit 5: Comprehensive guide to develop high quality Java applications / Boni García., 2017. – 545 p.
7. Parveen. A MANUAL TESTING GUIDE FOR BEGINNERS / Parveen.. – 28 p.
8. Jena A. Automated Software Testing: Foundations, Applications and Challenges / A. Jena, H. Das, D. Mohapatra., 2020. – 179 p.
9. Software Testing : An ISTQB-BCS Certified Tester Foundation guide / [eds.: B. Hambling, P. Morgan, A. Samaroo]., 2019. – 283 p. – (4).
10. Alsmadi I. Advanced Automated Software Testing / Izzat Alsmadi., 2012. – 288 c.
11. What is JBehave? [Electronic resource] – Access mode: <https://jbehave.org/>.
12. The 5th major version of the programmer-friendly testing framework for Java and the JVM [Electronic resource] – Access mode: <https://junit.org/junit5/>.
13. SERENITY – THAT FEELING WHEN YOU KNOW YOU CAN TRUST YOUR TESTS [Electronic resource] – Access mode: <https://serenity-bdd.info/>.

14. WHAT IS SELENIDE? [Electronic resource] – Access mode: <https://selenide.org/>.
15. TestNG [Electronic resource] – Access mode: <https://testng.org/doc/>.
16. Less Code, Less Maintenance, More Acceptance Testing [Electronic resource] – Access mode: <https://gauge.org/>.
17. Palani N. Software Automation Testing Secrets Revealed / Narayanan Palani., 2016. – 330 p.
18. Selenium automates browsers. That's it! [Electronic resource] – Access mode: <https://www.selenium.dev/>.
19. WebDriver [Electronic resource] – Access mode: <https://www.selenium.dev/documentation/webdriver/>.
20. UFT One [Electronic resource] – Access mode: <https://www.microfocus.com/en-us/products/uft-one/overview>.
21. An all-in-one test automation solution [Electronic resource] – Access mode: <https://www.katalon.com/>.
22. What is IBM® Rational® Functional Tester? [Electronic resource] – Access mode: <https://www.ibm.com/products/rational-functional-tester>.
23. Automated UI testing that covers you from device cloud to packaged apps [Electronic resource] – Access mode: <https://smartbear.com/product/testcomplete/overview/>.
24. Automation for Apps [Electronic resource] – Access mode: <https://appium.io/>.
25. Ranorex Studio: Functional UI Test Automation [Electronic resource] – Access mode: <https://www.ranorex.com/>.
26. Test automation for native or hybrid Android apps and the mobile web with Selendroid. [Electronic resource] – Access mode: <http://selendroid.io/>.
27. TIOBE Index for November 2021 [Electronic resource] – Access mode: <https://www.tiobe.com/tiobe-index/>.
28. Our world. Moved by Java. [Electronic resource] – Access mode: <https://www.oracle.com/java/moved-by-java/timeline/>.

29. Browsers & Mobile Devices for Live Testing [Electronic resource] – Access mode: <https://www.browserstack.com/list-of-browsers-and-platforms/live>.
30. Top IDE index [Electronic resource] – Access mode: <https://pypl.github.io/IDE.html>.
31. git--distributed-even-if-your-workflow-isnt [Electronic resource] – Access mode: <https://git-scm.com/>.
32. Download IntelliJ IDEA [Electronic resource] – Access mode: <https://www.jetbrains.com/idea/download/#section=windows>.

ДОДАТОК А
Текст програми

Додаток А.1 – Base

```

package common.base;

import common.DriverHelper;
import io.qameta.allure.Attachment;
import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;

public abstract class Base {
    @Attachment(type = "text/plain")
    public String saveOutput(String data) {
        return data;
    }

    @Attachment(value = "Output - {outPutName}", type =
"text/plain")
    public String saveOutput(String outPutName, String
data) {
        return data;
    }

    @Attachment(value = "Page screenshot -
{screenshotName}", type = "image/png")
    public byte[] saveScreenshot(String screenshotName) {
        return ((TakesScreenshot)
DriverHelper.getDriver()).getScreenshotAs(OutputType.BYTES
);
    }
}

```

Додаток А.2 – BaseBlock

```

package common.base;

import common.DriverHelper;
import common.utils.Log;
import io.qameta.allure.Step;
import org.openqa.selenium.support.PageFactory;
import ru.yandex.qatools.htmlelements.element.HtmlElement;
import
ru.yandex.qatools.htmlelements.loader.decorator.HtmlElementDecorator;
import
ru.yandex.qatools.htmlelements.loader.decorator.HtmlElementLocatorFactory;

public abstract class BaseBlock extends HtmlElement {

    protected DriverHelper h;
    protected DriverHelper hLong;
    protected DriverHelper hShort;

    public BaseBlock() {
        h = new DriverHelper(this, DriverHelper.TIMEOUT_LONG);
        hLong = new DriverHelper(this, DriverHelper.TIMEOUT_LONG);
        hShort = new DriverHelper(this, DriverHelper.TIMEOUT_SHORT);
        HtmlElementLocatorFactory factory = new
HtmlElementLocatorFactory(DriverHelper.getDriver());

```

```
PageFactory.initElements(new HtmlElementDecorator(factory), this);  
}
```

```
@Step("Is block displayed")  
public boolean isDisplayed() {  
    boolean block = h.isElementDisplayed(getWrappedElement());  
    Log.info("Verify is the "+ getName() +" block displays" );  
    return block;  
}  
}
```

Додаток А.3 – BasePage

```

package common.base;

import common.DriverHelper;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.PageFactory;
import
ru.yandex.qatools.htmlelements.loader.decorator.HtmlElementDecorator;
import
ru.yandex.qatools.htmlelements.loader.decorator.HtmlElementLocatorFactory;

public abstract class BasePage extends Base{

    protected DriverHelper h;
    protected DriverHelper hLong;
    protected DriverHelper hShort;

    public BasePage() {
        DriverHelper.waitForPageLoad();
        HtmlElementLocatorFactory factory = new
HtmlElementLocatorFactory(DriverHelper.getDriver());
        PageFactory.initElements(new HtmlElementDecorator(factory), this);
        hShort = new DriverHelper(DriverHelper.TIMEOUT_SHORT);
        h = new DriverHelper(DriverHelper.TIMEOUT_DEFAULT);
        hLong = new DriverHelper(DriverHelper.TIMEOUT_LONG);
    }

```

```
}
```

```
public void switchToFrame(WebElement element) {  
    DriverHelper.getDriver().switchTo().frame(element);  
}
```

```
public void switchToDefaultContent() {  
    DriverHelper.getDriver().switchTo().defaultContent();  
}
```

```
public void waitForPageLoad(){  
    DriverHelper.waitForPageLoad();  
}  
}
```

Додаток А.4 – BaseTest

```
package common.base;

import common.utils.Utills;
import common.utils.listeners.ScreenshotListener;
import common.utils.listeners.TestLogs;
import org.openqa.selenium.Dimension;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebDriverException;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeOptions;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.firefox.FirefoxOptions;
import org.openqa.selenium.remote.LocalFileDetector;
import org.openqa.selenium.remote.RemoteWebDriver;
import org.testng.annotations.*;

import java.net.MalformedURLException;
import java.net.URL;
import java.util.HashMap;
import java.util.Map;

@Listeners({ScreenshotListener.class, TestLogs.class})
public abstract class BaseTest extends Base{
```

```
private static ThreadLocal<WebDriver> driverPool = new ThreadLocal<>();
```

```
private final String GRID_URL = "grid-url";
```

```
private URL host;
```

```
private Map<String, Object> capabilities = new HashMap<>();
```

```
private ChromeOptions chromeOptions = new ChromeOptions();
```

```
private FirefoxOptions fireFoxOptions = new FirefoxOptions();
```

```
public void initChromeDriver(){
```

```
    if (Utils.isEnvironmentRemote()){
```

```
        //init Grid + remote web Driver
```

```
        initGridConnection();
```

```
        RemoteWebDriver remoteWebDriver = new RemoteWebDriver(host,  
chromeOptions);
```

```
        remoteWebDriver.setFileDetector(new LocalFileDetector());
```

```
        driverPool.set(remoteWebDriver);
```

```
    }else {
```

```
        System.setProperty("webdriver.chrome.driver",
```

```
"src\\test\\resources\\chromedriver.exe");
```

```
        driverPool.set(new ChromeDriver(chromeOptions));
```

```
    }
```

```
}
```

```

private void initFirefoxDriver() {
    if (Utils.isEnvironmentRemote()) {
        //init Grid + remote web Driver
        initGridConnection();
        RemoteWebDriver driver = new RemoteWebDriver(host,
fireFoxOptions);
        driver.setFileDetector(new LocalFileDetector());
        driverPool.set(driver);
    } else {
        System.setProperty("webdriver.gecko.driver", "/geckodriver.exe");
        driverPool.set(new FirefoxDriver((fireFoxOptions)));
    }
}

```

```

public void initGridConnection(){
    host = null;
    try {
        host = new URL(GRID_URL);
    }catch (MalformedURLException e){
        e.printStackTrace();
    }
}

```

```

public static WebDriver getDriver(){
    return driverPool.get();
}

```

```

@BeforeTest(alwaysRun = true)
@Parameters("Browser")
public void setUp(@Optional("chrome") String browser) {
    if (browser.equals("chrome")) {
        initChromeDriver();
    } else if (browser.equals("firefox")) {
        initFirefoxDriver();
    }
    WebDriver driver = getDriver();
    driver.manage().window().setSize(new Dimension(1920, 1080));
}

```

```

@AfterTest(alwaysRun = true)
public void quit() {
    try {
        driverPool.get().close();
        driverPool.get().quit();
    } catch (WebDriverException e) {
        e.printStackTrace();
    }
}
}

```

Додаток А.5 – BaseBrowserStack

```

package common.baseMobile;

import common.DriverHelper;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.remote.RemoteWebDriver;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Parameters;
import java.net.MalformedURLException;
import java.net.URL;

public class BaseBrowserStack extends BaseMobile {

    protected WebDriver driver;
    protected DriverHelper driverHelper;
    protected String device;

    @BeforeTest
    @Parameters({«browserName»,»device»,»os_version»})
    public void setCapabilitiesAndInitDriver(String browserName, String
device, String os_version) throws MalformedURLException {
        DesiredCapabilities caps = new DesiredCapabilities();
        this.device = device;
    }

```

```

        String                buildName                =
System.getenv("BROWSERSTACK_BUILD_NAME");
        String username = System.getenv("BROWSERSTACK_USERNAME");
        String                accessKey                =
System.getenv("BROWSERSTACK_ACCESS_KEY");

```

```

        caps.setCapability("browserName", browserName);
        caps.setCapability("device", device);
        caps.setCapability("realMobile", "true");
        caps.setCapability("os_version", os_version);
        caps.setCapability("name", "Test for " + device + " version " +
os_version); // test name
        caps.setCapability("build", buildName); // CI/CD job or build name

```

```

        String url = "https://" + username + ":" + accessKey +
"@hub.browserstack.com/wd/hub";

```

```

        System.out.println(url);
        driver = new RemoteWebDriver(new URL(
            /*getUrl()*/url
        ), caps);
        driverPool.set(driver);
    }

```

```
@AfterTest  
public void quit(){  
    driver.quit();  
}  
}
```

Додаток А.6 – BaseMobile

```
package common.baseMobile;
```

```
import org.openqa.selenium.WebDriver;
```

```
import java.io.FileInputStream;
```

```
import java.io.FileNotFoundException;
```

```
import java.io.IOException;
```

```
import java.util.Properties;
```

```
public class BaseMobile {
```

```
    protected static ThreadLocal<WebDriver> driverPool = new  
ThreadLocal<>();
```

```
    public static String getUsername(){  
        return readPropertiesFile().getProperty("name");  
    }
```

```
    public static String getPass(){  
        return readPropertiesFile().getProperty("pass");  
    }
```

```

public static String getSiteURL(){
    return readPropertiesFile().getProperty("url");
}

```

```

public static String getUrl(){
    return "https://" + getUsername() + ":" + getPass() +
"@hub-cloud.browserstack.com/wd/hub";
}

```

```

public static Properties readPropertiesFile(){
    FileInputStream fis = null;
    Properties properties = new Properties();
    try {
        fis = new
FileInputStream("D:\\Project_PPC\\ClickCease_BrowserStack\\src
\\test\\resources\\browserStack.properties");
        properties.load(fis);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }

    return properties;
}

```

```
public static WebDriver getDriver(){  
    return driverPool.get();  
}  
  
}
```

Додаток А.7 – BasePageMobile

```

package common.baseMobile;

import common.DriverHelper;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.PageFactory;
import
ru.yandex.qatools.htmlelements.loader.decorator.HtmlElementDecorator;
import
ru.yandex.qatools.htmlelements.loader.decorator.HtmlElementLocatorFactory;

public class BasePageMobile {

    public DriverHelper h;
    public DriverHelper hMinTimeout;
    public DriverHelper hMaxTimeout;

    public BasePageMobile() {
        DriverHelper.waitForPageLoad();
        HtmlElementLocatorFactory factory = new
HtmlElementLocatorFactory(DriverHelper.getDriver());
        PageFactory.initElements(new HtmlElementDecorator(factory), this);
        h = new DriverHelper(DriverHelper.TimeOutMob.DEFAULT);
        hMinTimeout = new DriverHelper(DriverHelper.TimeOutMob.SHORT);
        hMaxTimeout = new DriverHelper(DriverHelper.TimeOutMob.LONG);
    }
}

```

```
}
```

```
public void switchToFrame(WebElement element) {  
    DriverHelper.getDriver().switchTo().frame(element);  
}
```

```
public void switchToDefaultContent() {  
    DriverHelper.getDriver().switchTo().defaultContent();  
}  
}
```

Додаток А.8 – DriverHelper

```
import common.base.BaseTest;
import common.baseMobile.BaseMobile;
import common.utils.Log;
import common.utils.Utills;
import io.qameta.allure.Step;
import org.openqa.selenium.*;
import org.openqa.selenium.interactions.Actions;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;

import java.util.List;

public class DriverHelper {

    public static final int TIMEOUT_SHORT = 10;
    public static final int TIMEOUT_LONG = 60;
    public static final int TIMEOUT_DEFAULT = 30;

    private SearchContext searchContext;
    private JavascriptExecutor js;
    private WebDriverWait webDriverWait;
```

```

public DriverHelper(SearchContext searchContext, int timeout){
    this.searchContext = searchContext;
    webDriverWait = new WebDriverWait(getDriver(), timeout);
    js = (JavascriptExecutor) getDriver();
}

```

```

public static WebDriver getDriver(){
    return BaseTest.getDriver();
}

```

```

public static WebDriver getDriverMobile(){
    return BaseMobile.getDriver();
}

```

```

public DriverHelper(int timeout) {
    this(getDriver(), timeout);
}

```

```

public DriverHelper(TimeOutMob timeout) {
    this(getDriverMobile(), timeout.time);
}

```

```

DriverHelper() {
    this(TIMEOUT_DEFAULT);
}

```

```
}
```

```
public static void waitForPageLoad(){
    Log.info("Waiting for page load");
    WebDriverWait webDriverWait = new WebDriverWait(getDriver(), 30);
    webDriverWait.until(wait ->
        String.valueOf(((JavascriptExecutor) getDriver()).executeScript("return
document.readyState")))
        .equals("complete"));
}
```

```
@Step("Wait until url contains - {text}")
public static boolean waitForUrlContains(String text){
    Log.info("Waiting while page URL contains : " + text);
    WebDriverWait webDriverWait = new WebDriverWait(getDriver(), 30);
    webDriverWait.until(ExpectedConditions.urlContains(text));
    return getDriver().getCurrentUrl().contains(text);
}
```

```
public WebElement waitForVisibility(WebElement element){
    Log.info("Waiting for visibility of" + element);
    return webDriverWait.until(ExpectedConditions.visibilityOf(element));
}

public WebElement waitForVisibility(By locator){
    Log.info("Waiting for visibility of" + locator);
    return
```

```
webDriverWait.until(ExpectedConditions.visibilityOfElementLocated(locator));
}
```

```
@Step("Open url -{url}")
public static void openUrl(String url){
    getDriver().get(url);
    waitForPageLoad();
    Log.info("The " + url + " is opened");
}
```

```
@Step("Open url -{url}")
public static <T> T openUrl(String url, Class <T> clazz){
    getDriver().get(url);
    waitForPageLoad();
    Log.info("The " + url + " is opened");
    return (T) clazz;
}
```

```
/**
 * Refreshing page
 */
```

```
@Step("Refreshing page")
public static void refreshPage(){
    getDriver().navigate().refresh();
}
```

```

    Log.info("The page is refreshed");
}

```

```

@Step("Refreshing page and wait until load")
public static void refreshPageAndWait(){
    refreshPage();
    waitForPageLoad();
}

```

```

public void scrollToElement(WebElement element){
    js.executeScript("argument[0].scrollIntoView(true)", element);
    Log.info("Scrolled to the element" + element);
}

public void scrollToElement(By locator){
    WebElement element = findElement(locator);
    js.executeScript("argument[0].scrollIntoView(true)", element);
    Log.info("Scrolled to the element" + element);
}

```

```

public void clearCookiesAndCached(){
    getDriver().get("chrome://settings/clearBrowserData");
    Utils.delayMillisec(5000);
    Actions actions = new Actions(getDriver());
    for (int i = 0; i < 7; i++) {
        actions.sendKeys(Keys.TAB).build().perform();
        Utils.delayMillisec(1000);
    }
}

```

```

    }
    actions.sendKeys(Keys.ENTER).build().perform();
    Utils.delayMillisec(10000);
    Log.info("Cleared the cookies and cache");
}

```

```

public void sendESC(){
    new Actions(getDriver()).sendKeys(Keys.ESCAPE).perform();
    Log.info("The ESC button is clicked");
}

```

```

public WebElement findElement(By xpath){
    return getDriver().findElement(xpath);
}

```

```

public List<WebElement> findElements(By xpath){
    List<WebElement> elements = getDriver().findElements(xpath);
    Log.info("Total amount of founded elements: " + elements.size());
    return elements;
}

```

```

public boolean isElementDisplayed(WebElement element){
    try {
        return waitForVisibility(element).isDisplayed();
    } catch (WebDriverException e){

```

```

        Log.info("The element" + element + " isn't displayed");
        return false;
    }
}

```

```

public boolean isElementDisplayed(By locator){
    try {
        return waitForVisibility(locator).isDisplayed();
    } catch (WebDriverException e){
        Log.info("The element" + locator + " isn't displayed");
        return false;
    }
}

```

```

public void clickWebElement(WebElement element){

    webDriverWait.until(ExpectedConditions.elementToBeClickable(element));
    element.click();
    Log.info("The " + element + " is clicked");
}

```

```

public void clickWebElement(By locator){
    WebElement element
    webDriverWait.until(ExpectedConditions.elementToBeClickable(locator));
    clickWebElement(element);
}

```

=

```

public void typeIntoElement(WebElement element, String text){
    element.sendKeys(text);
    Log.info("The text is inserted to the element " + element);
}

```

```

public void getElementAttribute(WebElement element, String attribute){
    waitForVisibility(element);
    try {
        element.getAttribute(attribute);
    } catch (WebDriverException e) {
        Log.error("The attribute isn't founded " + attribute + " | " + element );
    }
}

```

```

public enum TimeOutMob{
    SHORT(10),
    LONG(60),
    DEFAULT(30);

```

```

    public int time;

```

```

    TimeOutMob(int time){
        this.time=time;
    }

```

}
}
}

Додаток А.9 - temp.xml

```
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >  
<suite name="12">  
  <test name="1">  
    <test name="1">  
      <classes>  
        <class name="InnerTest"/>  
      </classes>  
    </test>  
  </suite>
```

Додаток А.10 - temp _mobile.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Single">
  <test name="iPhone">
    <parameter name="browserName" value="iPhone"/>
    <parameter name="device" value="iPhone 12"/>
    <parameter name="os_version" value="14"/>
    <classes>
      <class name="SignUpAndOnboardingTest"/>
    </classes>
  </test>
  <test name="Android">
    <parameter name="browserName" value="Android"/>
    <parameter name="device" value="Samsung Galaxy S21"/>
    <parameter name="os_version" value="11.0"/>
    <classes>
      <class name="SignUpAndOnboardingTest"/>
    </classes>
  </test>
</suite>
```

ДОДАТОК Б
Презентація

**Національний університет «Запорізька політехніка»
Кафедра програмних засобів
Атестаційна робота магістра**

**ДОСЛІДЖЕННЯ МЕТОДІВ ТА ПРОГРАМНИХ
ЗАСОБІВ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ САЙТІВ
НА КОМП'ЮТЕРАХ І МОБІЛЬНИХ ПРИСТРОЯХ**

2021

Науковий керівник -

Завідувач кафедри програмних засобів

Доктор технічних наук

Професор Субботін Сергій Олександрович

Підготував студент 2-го курсу КНТ-210м

Тимофєєв Єгор Григорович

1

- **Метою** кваліфікаційної роботи є дослідження методів та засобів автоматизації сайтів, а також створення програми, яка буде слугувати, тестовим фреймворком, за допомогою якого можна легко почати автоматизоване тестування на комп'ютерах чи мобільних пристроях.
- **Об'єктом** дослідження - є процес тестування програмних продуктів
- **Предмет** дослідження - є засоби та методи автоматизації процесу тестування програмних продуктів.
- **Завдання**
Дослідити інструменти та засоби автоматизації
Виконати аналіз інструментів та засобів автоматизації
Створити фреймворк для автоматизації процесу тестування.

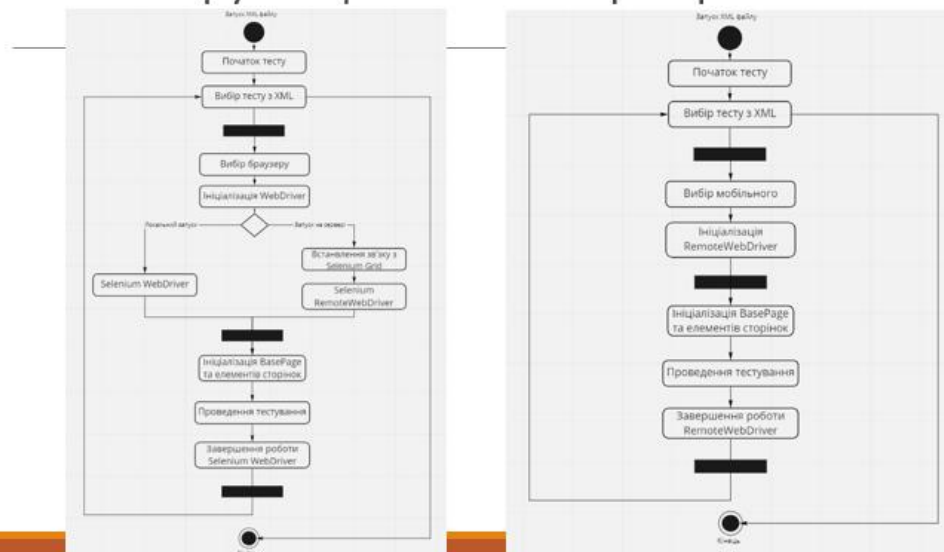
2

Огляд існуючих технологій

	Вартість	Платформи для тестування	Необхідність знання мови для написання тестів	Використання мов програмування
Ranorex	Платна	Комп'ютерні та мобільні платформи	Не потребує	VB .Net, C#
TestComplete	Платна	Комп'ютерні та мобільні платформи	Не потребує	JavaScript, Python, VBScript, JScript, Delphi, C++, C#
KatalonStudio	Платна (Є безкоштовна версія)	Комп'ютер та мобільні платформи	Не потребує	Java, Groovy

3

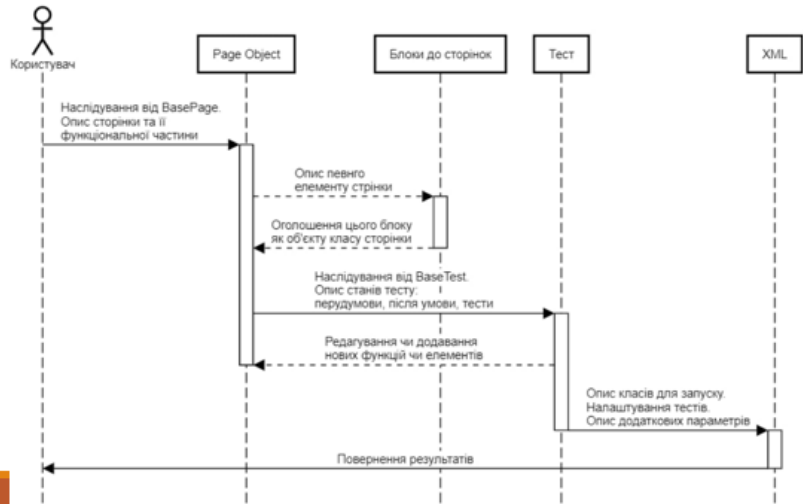
Схеми функціонування програми



4

Схеми роботи з програмою

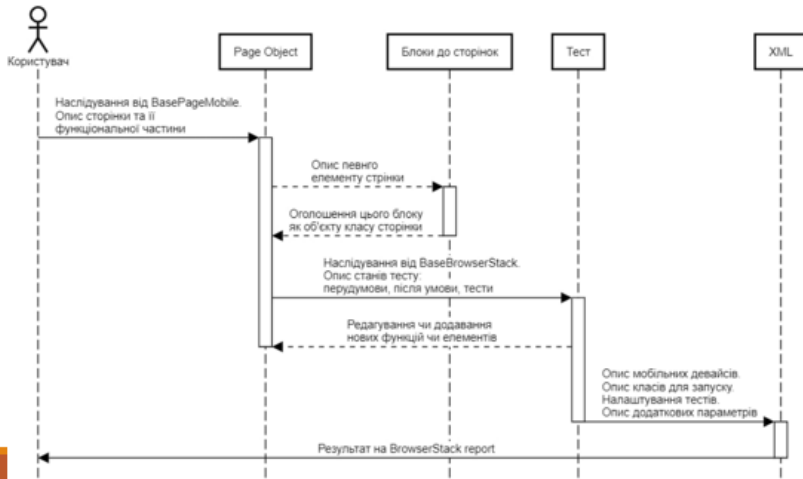
Схема написання тестів для комп'ютера



5

Схеми роботи з програмою

Схема написання тестів для телефону



6

Вибір мови програмування

	Типізація даних	Продуктивність	Легкість читання мови
Java	Суворі типізація	Виконує своєчасну компіляцію всіх програм	Є більш важкою для читання та потребує досконалішого знання мови
Python	Динамічна типізація	Компілює код під час виконання	Є простою для зрозуміння, але можуть виникати проблеми при пошуку помилок за рахунок табуляції

7

Порівняння фреймворків для тестування

Властивість Фреймворк	Паралелізація (паралельний запуск тестів)	Життєвий цикл	Параметризація тестів
JBehave	Присутня	Є BDD (розробка проходить по іншому)	Відсутня
JUnit	Присутня	Великий вибір анотацій	Присутня
Serenity	Присутня	Є BDD (розробка проходить по іншому)	Відсутня
Selenide	Присутня	Великий вибір анотацій	Відсутня
TestNG	Присутня	Великий вибір анотацій	Присутня
Gauge	Присутня (але з проблемами)	Є більш орієнтованим на читабельність кода	Відсутня

8

Порівняння інструментів для тестування Web-додатків

Властивість Інструмент	Операційна система	Мови програмування, які підтримує інструмент	Можливість використання неперервної інтеграції	Тип ліцензії та вартість
Selenium	Windows, Linux, OS X	Bci	Присутня	Безкоштовна
UFT	Windows	VBScript	Присутня	Платна
Katalon Studio	Windows, Linux, OS X	Java/Groovy	Присутня	Безкоштовна
IBM RFT	Windows, Linux	Java	Присутня	Безкоштовна
TestComplete	Windows	JS, python, VBScript, JScript	Присутня	Безкоштовна

9

Порівняння інструментів автоматизації мобільних Web-додатків

Властивість Інструмент	Операційна система	Мови програмування, які підтримує фреймворк	Паралелізація (паралельний запуск тестів)
Appium	Android, iOS, WebView	Bci	Дозволяє паралелізацію
Detox	Android, iOS, WebView	JavaScript	Лише 1 додаток
Ranorex	Android, iOS, WebView	C#	Дозволяє паралелізацію
Selendroid	Android, WebView	Java	Лише 1 додаток

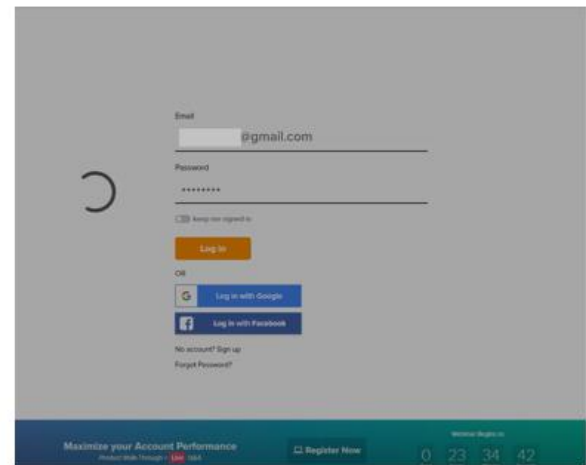
10

Приклад тесту для входу в обліковий запис

```

35 @AfterMethod
36 public void clearCookies(){
37     new DriverHelper().clearCookiesAndCached_NEW();
38 }
39
40 @Test(testName = "C705144",
41       description = "Make sure that the user can Log In to the site", priority = 1)
42 @TestLink(value = "705144")
43 public void loginTest(){
44     Allure.getLifecycle().updateTestCase(testResult ->
45     {
46         testResult.setName(testResult.getDescription());
47         testResult.setHistoryId(testResult.getHistoryId() + "roleID");
48     });
49     mainClickCeasePage = DriverHelper.openClickCeasePageStage();
50     loginPage = main.clickLogin();
51     user = DriverManager.getInstance().getById(" ");
52     loginPage.typeUsername(user);
53     loginPage.typePassword(user);
54     accountOverviewPage = loginPage.clickLoginButton();
55     Assert.assertTrue("The page isn't loaded", accountOverviewPage.isPageDisplayed());
56 }
57
58 @Test(testName = "C705145",
59       description = "Make sure that the user can SIGN UP", priority = 2)
60 @TestLink(value = "705145")
61 // todo
62 public void signTest(){
63     Allure.getLifecycle().updateTestCase(testResult ->
64     {
65         testResult.setName(testResult.getDescription());
66         testResult.setHistoryId(testResult.getHistoryId() + "roleID");
67     });
68     SoftAssert softAssert = new SoftAssert();
69 }

```



11

Приклад PageFactory класу

```

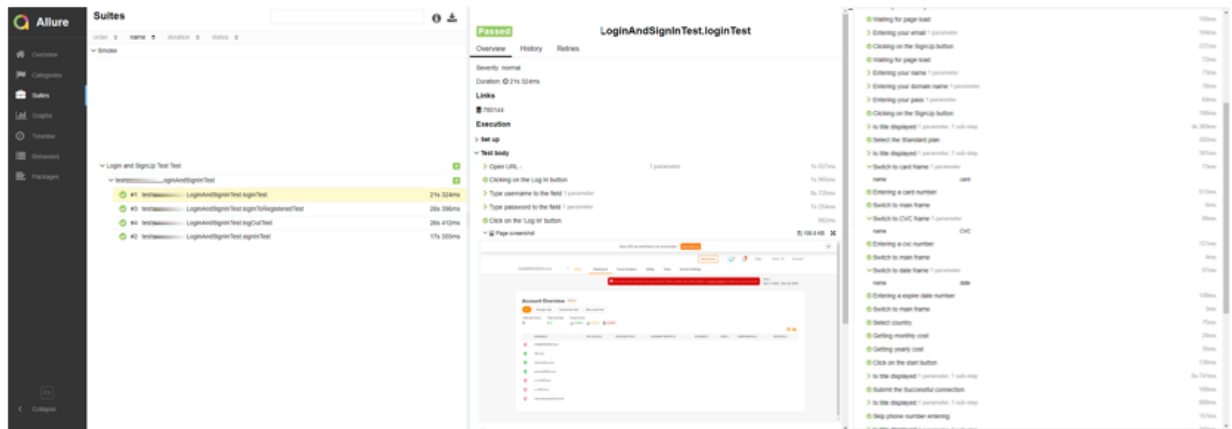
@Name("Onboarding")
@FindBy(xpath = "//div[@class=\"onboarding-container\"]")
public class OnboardingPage extends BaseBlock {

    @FindBy(xpath = "//h3")
    private WebElement titleH3;
    @FindBy(xpath = "//div[@class=\"tracking-code-step-mobile\"]//h3")
    private WebElement titleTrackMobileH3;
    @FindBy(xpath = "//h1")
    private WebElement titleH1;
    @FindBy(xpath = "//h4")
    private WebElement subTitleH4;
    @FindBy(xpath = "//div[@class=\"tracking-code-step-mobile\"]//h4")
    private WebElement subTitleTrackMobileH4;
    @FindBy(xpath = "//h3")
    private WebElement subTitleH3;
    @FindBy(xpath = "//h5[@class=\"back-to-dash\"]")
    private WebElement goToDashboardButton;
}

```

12

Демонстрація звіту



13

Висновки

- виконано класифікацію та порівняльний аналіз інструментів та засобів автоматизованого тестування;
- обрано найефективніші інструменти;
- спроектовано та створено програмний продукт для автоматизованого тестування

14