ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «РОЗРОБКА РУЧНИХ ЗАСОБІВ ТЕСТУВАННЯ ВЕБСАЙТІВ»

на здобуття освітнього ступеня бакалавра
зі спеціальності 122 Комп'ютерні науки
освітньо-професійної програми Штучний інтелект

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Андрій ЛЯЛЮШКО

Виконав: Андрій ЛЯЛЮШКО
здобувач вищої освіти
група ШІД-41

Керівник: Юлія БЕРЕЗОВСЬКА
Доктор філософії

Рецензент:
_____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Штучного інтелекту

Ступінь вищої освіти Бакалавр

Спеціальність 122 Комп'ютерні науки

Освітньо-професійна програма Штучний інтелект

ЗАТВЕРДЖУЮ
Завідувач кафедрою Штучного інтелекту

_____ Ольга ЗІНЧЕНКО
«_____» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Лялюшку Андрію Миколайовичу

1. Тема кваліфікаційної роботи: Розробка ручних засобів тестування вебсайтів

керівник кваліфікаційної роботи Юлія БЕРЕЗОВСЬКА, доктор філософії

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» 02.2024р. № 36

2. Строк подання кваліфікаційної роботи 31 травня 2024р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, види ручних тестувань.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
   1. Дослідження побудови ручного та автоматизованого тестувань.
   2. Аналіз та порівняння технологій тестувань.
   3. Розробка вимог для функціональності сайтів.

5. Перелік графічного матеріалу: *презентація*
   1. Аналіз існуючих підходів до ручного тестування вебсайтів
   2. Розробка комплексного плану тестування, включаючи визначення критеріїв успішності

3. Виконання ручного тестування автоматизованими скриптами на практичних прикладах
4. Оцінка ефективності ручного тестування та рекомендації щодо його оптимізації.

6. Дата видачі завдання 27 лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	27.02-08.03.24	Виконано
2	Визначення цілей та завдань дослідження	09.03-19.03.24	Виконано
3	Розробка методології тестування	20.03-30.03.24	Виконано
4	Вибір інструментів та технік тестування	31.03-06.04.24	Виконано
5	Проведення експериментального тестування	07.04-16.04.24	Виконано
6	Аналіз результатів тестування	17.04-26.04.24	Виконано
7	Оформлення роботи: вступ, висновки, реферат	27.04-01.05.24	Виконано
8	Розробка демонстраційних матеріалів	02.05-27.05.24	Виконано

Здобувач вищої освіти _____ Андрій ЛЯЛЮШКО

Керівник
кваліфікаційної роботи _____ Юлія БЕРЕЗОВСЬКА

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 71 стор., 5 табл., 64 рис., 17 джерел.

Мета роботи – розробити рекомендації та методичні вказівки щодо ручного тестування вебсайтів.

Об’єкт дослідження – процеси та методики ручного тестування вебсайтів.

Предмет дослідження – розробка та впровадження ручних засобів для тестування вебсайту .

У даній роботі розглянуто основи та методи ручного тестування вебсайту. Проаналізовано переваги та недоліки ручного тестування та автоматизованих методів тестування. Описано різноманітні типи тестування, які можна виконати вручну, включаючи функціональне тестування, тестування інтерфейсу користувача, тестування на сумісність і тестування безпеки.

Розроблено автоматизовану структуру для тестування, що буде легко працювати над своїм проєктом на всіх платформах, щоб сприяти його вдосконаленню та одержувати найвищу якість.

КЛЮЧОВІ СЛОВА: ШТУЧНИЙ ІНТЕЛЕКТ, ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ, МАШИННЕ НАВЧАННЯ, ОБРОБКА ПРИРОДНИХ МОВ, КОМП’ЮТЕРНИЙ ЗІР, ІНТЕРНЕТ РЕЧЕЙ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ПЛАТФОРМА, ІНФРАСТРУКТУРА, ІНТЕРФЕЙС ПРОГРАМУВАННЯ ДОДАТКІВ.

ЗМІСТ

ВСТУП

Вебсайти відіграють важливу роль у бізнесі, освіті та комунікації. Для успіху онлайн проєкту важливими є надійність, безпека та зручність використання сайту.

Ручний аналіз включає тестування функціональності, визначення помилок посилань та форм, оцінку інтуїтивності інтерфейсу, аналіз стабільності під навантаженням, виявлення вразливостей та тестування впливу нових змін на функціональність. Кожен етап вимагає ретельної роботи, адже від якості відгуків залежить подальша ефективність та успіх роботи сайту. Ручний підхід дозволяє краще зрозуміти потреби користувачів та виявити проблеми, які можуть бути пропущені під час автоматичного аналізу. Це особливо важливо для вебсайту, де персональний досвід користувача є пріоритетом.

Наукове дослідження відображає відповідність поточним проблемам та викликам, а також демонструє, яким чином може сприяти розвитку наукової сфери. Оскільки вебсайти є важливими інструментами для бізнесу, освіти, спілкування та багатьох інших галузей, то важливо застосовувати розробки інструментів ручного тестування. Високоякісне тестування вебсайту гарантує, що він є надійним, безпечним та простим у використанні. А це важливо для успіху проєкту будь-якої компанії.

Також, важливим етапом у процесі розробки є тестування на місці, тому що підтверджує високу якість продукту та його відповідність технічним вимогам і стандартам. Даний процес буде корисним не лише для пошуку помилок та недоліків, але й для підвищення продуктивності, забезпечення безпеки та покращення взаємодії з користувачем.

Застосовуючи перевірку досягається правильність роботи основних функцій ресурсу, виявлення помилкових посилань, перевірки користувачів, авторизацій та реєстрацій.

За допомогою тестування є можливість оцінити зручність використання вебсайту, включаючи інтуїтивність інтерфейсу користувача і легкість навігації. А

також, використавши тестування продуктивності проаналізувати результативність та стабільність сайту під високим навантаженням.

Для виявлення потенційних уразливостей та захисту від зовнішніх загроз використовується тестування безпеки. А застосувавши регресійне тестування є можливість переконатися, що нові зміни не порушують наявну функціональність.

Кожен із цих етапів вимагає уважності та детального підходу, адже від якості тестування залежить майбутня надійність та успіх сайту. Тестування вручну дозволяє краще зрозуміти взаємодію користувачів та виявити проблеми, які неочевидні за допомогою автоматичного тестування. Даний підхід є особливо важливим для вебсайту, де важливий персональний досвід користувача.

1 МЕТОДИКИ ТА ПРАКТИКИ РУЧНОГО ТЕСТУВАННЯ ВЕБСАЙТІВ: ВИВЧЕННЯ ТА АНАЛІЗ

1.1 Огляд сучасних методик ручного тестування вебсайтів

Огляд сучасних методів ручного тестування вебсайту включає аналіз різних підходів, які використовуються для забезпечення якості та надійності вебресурсу.

Ручне тестування залишається важливим елементом процесу веброзробки, надаючи тестерам гнучкість реагувати на несподівані проблеми та високий рівень уваги до деталей.

1. Функціональне тестування: перевірка сайту на відповідність встановленим вимогам і специфікаціям. Тестери перевіряють усі функції вебсайту, включаючи форми, посилання, бази даних, інтерактивні модулі тощо. Цей тип тестування також передбачає перевірку взаємодії різних модулів вебсайту, щоб переконатися, що вони працюють разом правильно;

2. Перевірка зручності використання: оцінка зручності використання вебсайту включає тестування навігації, читабельності вмісту та доступності для людей з обмеженими можливостями. Важливою частиною тестування зручності використання є збір.

3. Тестування сумісності: перевірка працездатності вебсайту на різних браузерах, операційних системах і пристроях. Тестування сумісності зазвичай перевіряє, як вебсайт виглядає в різних версіях браузера, щоб уникнути проблем з відображенням.

4. Тестування продуктивності: визначає здатність вебсайту витримувати високі навантаження, швидкість завантаження сторінки та ефективність коду. Окрім швидкості завантаження, тестування продуктивності також оцінює час відповіді сервера та оптимізацію бази даних.

5. Тестування безпеки: перевіряє вебсайт на наявність вразливостей і потенційних загроз, таких як SQL-ін'єкції, XSS-атаки та неналежна обробка сесій. Щоб захистити свій вебсайт, важливо регулярно оновлювати протоколи безпеки та проводити тестування на проникнення.

6 . Регресійне тестування: це набір тестів, спрямованих на виявлення помилок в модулях програми, які вже були протестовані. Це робиться не для перевірки відсутності помилок, а для того, щоб знайти і виправити помилки регресії.

Кожен із цих методів вимагає ретельного планування та виконання, а також постійного спілкування між розробниками та тестувальниками для забезпечення найвищої якості кінцевого продукту. Також важливо враховувати деталі вашої хмарної інфраструктури. Це пояснюється тим, що хмарна інфраструктура може впливати на процес тестування та вимагати спеціального підходу, особливо у випадку гібридних або публічних хмар. Як і у випадку з приватними хмарами, забезпечення безпеки даних і повний контроль над інфраструктурою є важливими аспектами, які слід враховувати під час ручного тестування вебсайту.

1.2 Аналіз ефективності різних методик тестування

Аналіз ефективності різних методів тестування є ключем до забезпечення якості програмного забезпечення. Ефективність тесту можна визначити як ступінь досягнення цілей з мінімальними затратами зусиль і ресурсів.

Розглянемо три основні методи: функціональне тестування, тестування продуктивності та автоматизоване тестування.

Функціональне тестування перевіряє, чи програмне забезпечення відповідає визначеним вимогам і специфікаціям. Це включає тестування окремих функцій і процесів у системі. Цей тип тестування важливий для виявлення помилок, які можуть вплинути на роботу програми.

Тестування продуктивності оцінює продуктивність програмного забезпечення за певних умов, особливо під високим навантаженням. Це включає тестування навантаження, стрес-тестування та тестування стабільності. Це гарантує надійну роботу програмного забезпечення для кінцевого користувача.

Автоматизоване тестування використовує спеціалізоване програмне забезпечення для запуску тестів і порівняння фактичних і очікуваних результатів. Автоматизація значно розширює сферу тестування та підвищує точність, але

розробка сценаріїв тестування потребує початкових інвестицій. Для порівняння цих методів можна використовувати такі критерії, як вартість, час, покриття, точність і використання ресурсів. Нижче наведено порівняння цих видів у таблиці 1.1.

Таблиця 1.1 – Порівняльна таблиця видів тестування

Вид тестування	Навантаженість	Час	Обсяг покриття	Точність	Ресурси
Функціональне тестування	Низька	Високий	Середній	Висока	Людські
Тестування продуктивності	Середня	Середній	Низький	Середня	Технічні
Автоматизоване тестування	Висока	Низький	Високий	Висока	Технічні

Кожна методика має переваги і недоліки, і вибір залежить від конкретної мети тестування.

Функціональне тестування є ефективним у забезпеченні відповідності функцій вимогам, тестування продуктивності забезпечує стабільність під навантаженням, а автоматизоване тестування забезпечує регулярне та широке тестування без особливих зусиль людини.

Вибір методів тестування повинен ґрунтуватися на аналізі вимог проєкту та наявних ресурсів.

1.3 Вивчення основних інструментів, що використовуються в ручному тестуванні

Основні інструменти, які використовуються в ручному тестуванні:

1. Чек-листи та тест-кейси. Це основні інструменти ручного тестування, які допомагають тестувальникам відстежувати виконання тесту. Вони містять список умов або змінних, які тестувальники використовують для перевірки функціональності програмного забезпечення. Контрольні списки забезпечують

систематичний підхід до тестування та гарантують, що ви не пропустите жодного важливого тестового сценарію. Тестові випадки детально описують кроки, очікувані результати та умови кожного тесту.

2. Bug Tracking Tools. Інструменти відстеження помилок, такі як Jira та Bugzilla, використовуються для реєстрації, відстеження та керування помилками. Це дозволяє командам працювати разом, щоб виявляти й усувати помилки та аналізувати тенденції помилок.

3. Test Management Tools. Інструменти керування тестуванням, такі як TestRail і HP ALM, забезпечують центральне розташування для зберігання тестів, планів тестування, відстеження виконання тестів і звітів. Це допомагає координувати зусилля з тестування та забезпечує прозорість процесу тестування. Ці інструменти також можуть автоматизувати деякі аспекти тестування, наприклад: Приклади: надсилання повідомлень про статус тестування або створення звітів про помилки.

4. Exploratory Testing Tools. Інструменти дослідницького тестування, такі як Session Tester і Rapid Reporter, допомагають тестувальникам вести записи під час дослідницького тестування. Вони дозволяють швидко фіксувати спостереження, помилки та ідеї, які виникають під час тестування.

Також можна інтегрувати ці інструменти з іншими системами, щоб створити єдиний робочий процес для проєктів розробки програмного забезпечення. Надає звіти й аналітику, які допомагають приймати рішення щодо якості продукції.

У таблиці 1.2 наведено порівняльну характеристику найпопулярніших інструментів для тестування

Таблиця 1.2 – Порівняння популярних інструментів для тестування

Інструмент	Плюси	Мінуси
Selenium	Автоматизація тестування для різних браузерів; підтримка паралельного виконання тестів	Вимагає глибоких знань програмування; деякі тестові середовища вимагають складної конфігурації
TestLink	Централізоване управління тест-кейсами; інтеграція з іншими інструментами	Не інтуїтивно зрозуміло для нових користувачів; щоб його освоїти, потрібен час
LoadRunner	Ефективний для тестування продуктивності та навантаження; симуляція великої кількості користувачів	Може бути дорогим; важко освоїти та налаштувати
Bugzilla	Потужний для відстеження помилок; часто використовується в проєктах з відкритим кодом	Налаштування може бути складним; інтерфейс користувача не завжди інтуїтивно зрозумілий

Ці інструменти забезпечують більш ефективне та організоване пошукове тестування, дозволяючи тестувальникам зосередитися на пошуку неочікуваних проблем. Вони є ключем до ефективного ручного тестування та допомагають забезпечити якість і надійність програмного забезпечення.

На рисунку 1.1 наведено найкращі інструменти для тестування.

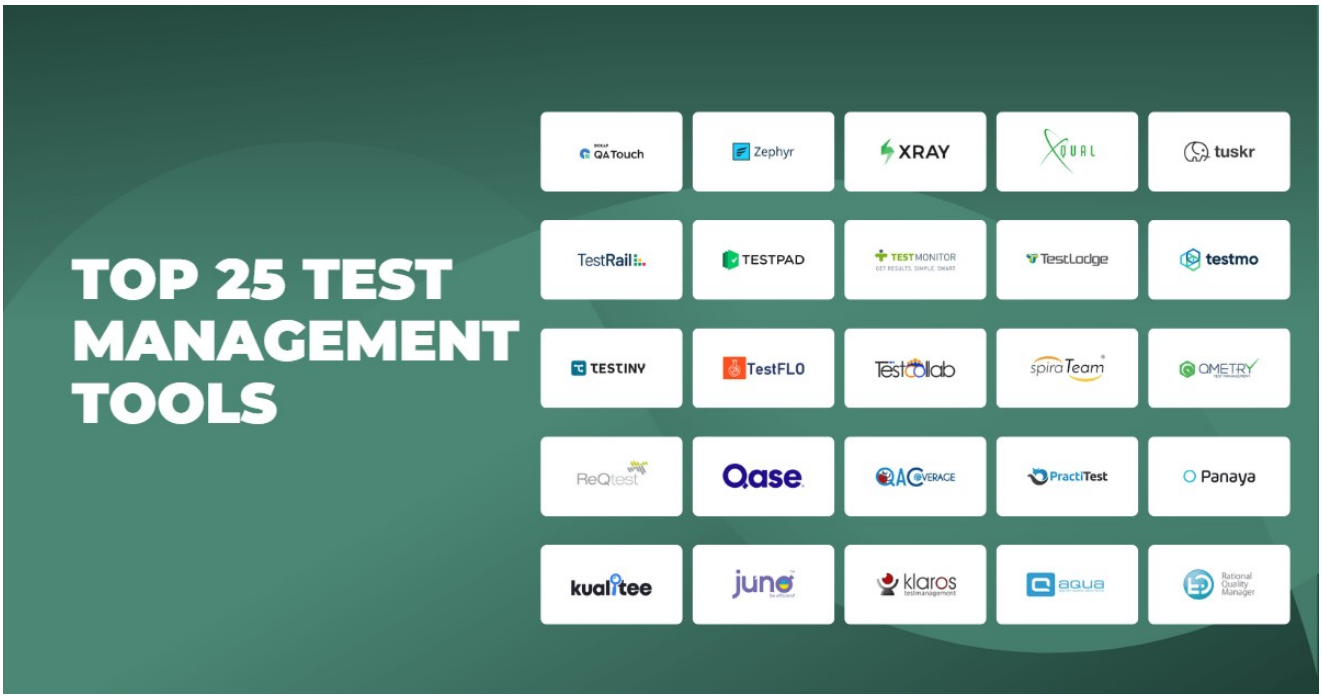


Рисунок 1.1 – 25 найкращих інструментів для тестування

Усі ці інструменти сприяють більш організованому та ефективному дослідницькому тестуванню, дозволяючи тестувальникам зосередитися на пошуку неочікуваних проблем, а також забезпечуючи якість і надійність тесту.

1.4 Розгляд практичних прикладів використання різних методик тестування

У даному підрозділі розглядаються основи та методи ручного тестування вебсайту. Аналізуються переваги та недоліки ручного тестування порівняно з автоматизованими методами. Описуються різні типи тестування, які можна виконати вручну, включаючи функціональне тестування, тестування інтерфейсу користувача, тестування на сумісність і тестування безпеки. Також, обговорюються інструменти та методи, які можна використовувати для підвищення ефективності ручного тестування, наприклад, контрольні списки, тестові випадки та варіанти використання.

Статичне тестування передбачає аналіз програмного коду, документації та інших артефактів програмування без виконання коду. Його метою є виявлення помилок на ранніх стадіях розробки, що економить час і ресурси.

Методи статичних випробувань:

1. Перегляд коду: процес перегляду коду програмного забезпечення іншими розробниками або тестувальниками для виявлення помилок, порушень стилю коду та інших проблем;

2. Перевірка стилю коду: аналіз коду на відповідність певним стандартам стилю коду, що допомагає забезпечити кращу читабельність коду та зручність обслуговування;

3. Статичний аналіз коду: автоматична перевірка свого коду за допомогою спеціальних інструментів, які допомагають виявити помилки, уразливості та порушення інструкцій щодо якості коду.

Важлива частина статичного тестування – тестування системних вимог. На цьому етапі вимоги аналізуються та перевіряються на можливі протиріччя, недосконалості та двозначності.

Динамічне тестування передбачає тестування програмного забезпечення шляхом його запуску. Мета полягає в тому, щоб виявити помилки в роботі програмного забезпечення та перевірити його функціональність.

Динамічні методи випробувань:

1. Модульне тестування: тестування окремих модулів або функцій програми;

2. Тестування інтеграції: перевірка взаємодії між різними модулями;

3. Тестування системи: тестування всієї системи в цілому;

4. Приймальні випробування: перевірка відповідності системи вимогам користувача.

Обидва методи мають свої переваги та недоліки (табл. 1.3). Статичне тестування дозволяє виявити помилки раніше, але не гарантує, що вони будуть виявлені повністю. Обидва методи важливі для забезпечення якості програмного забезпечення. Вибір методу залежить від конкретних вимог і обставин проєкту.

Таблиця 1.3 – Порівняльний аналіз статичного та динамічного тестування

Динамічне тестування	Виявлення помилок, пов’язаних зі змінними даних під час виконання. Перевірка працездатності програми в реальних умовах	Потрібен робочий код. Відтворює всі можливі сценарії взаємодії важко. Витрати на підготовку тестових даних і навчання.	Модульне тестування різних функцій. Інтеграційне тестування взаємодії між модулями. Тестування системи перевіряє всю систему
Статичне тестування	Виявляє помилки на ранніх стадіях розробки. Зменшує витрати на виправлення помилок пізніше. Покращує якість коду та документації	Нездатність виявити всі помилки (наприклад, помилки, пов’язані зі змінними даних під час виконання) потребує додаткового часу та зусиль для аналізу коду	Для перегляду коду використовується такий інструментт, як Code Review. Аналіз структури бази даних і схеми таблиць. Переконатися, що документ відповідає вимогам

Тому, в процесі розробки програмного забезпечення ручне та автоматизоване тестування відіграють ключову роль у забезпеченні якості продукту. Ручне тестування, хоча і займає багато часу, надає тестувальникам гнучкий метод виявлення помилок, особливо в інтерфейсі користувача та взаємодії з користувачем. Автоматизоване тестування, з іншого боку, ефективне

для повторюваних завдань і може значно прискорити процес тестування, особливо регресійне тестування.

Статичне тестування важливе для виявлення помилок на ранніх стадіях розробки, що запобігає ускладненням у подальшому процесі розробки. Воно включає перегляд коду, перевірку стилю коду та статичний аналіз коду, кожен з яких призначений для покращення якості коду та виявлення потенційних проблем до того, як вони з’являться у програмі.

Динамічне тестування, у свою чергу, дозволяє перевірити справжню поведінку програми під час виконання, виявляючи помилки, які не можуть виявити статичні методи. Методи динамічного тестування, такі як модульне тестування, інтеграційне тестування, системне тестування та приймальне тестування, дозволяють оцінити різні аспекти програмного забезпечення.

Вибір між ручним, статичним і динамічним тестуванням залежить від конкретних цілей і вимог проєкту. Ідеальним підходом є використання комбінації цих методів для досягнення найвищої якості програмного продукту. Важливо розуміти, що жоден метод не є універсальним і кожен метод має свої переваги та недоліки, які слід враховувати при плануванні тестування.

1.5 Розробка рекомендацій для впровадження ефективних методик ручного тестування

Ручне тестування є ключовим елементом процесу забезпечення якості програмного забезпечення. Воно дозволяє тестувальникам виявляти помилки на ранніх стадіях розробки, що може значно знизити витрати на виправлення помилок. Хоча інструменти автоматизації продовжують розвиватися, ручне тестування залишається незамінним, оскільки воно враховує людський фактор і забезпечує гнучкість для виявлення неочевидних помилок. Для ефективного ручного тестування необхідно мати чіткий план, який включає:

1. Розробка тестового випадку: докладний опис кроків, які слід виконати для тестування кожної функції;

2. Виконання тесту: ретельно виконувати тести та записувати результати;

3. Аналіз результатів: визначити помилки та визначити пріоритетність виправлень;

4. Обрати інструмент ручного тестування: важливо вибрати інструмент, який дозволить ефективно керувати тестовими випадками, збирати результати тестів і спілкуватися з командою. Це можуть бути системи управління проєктами, такі як JIRA чи Trello, або спеціальні інструменти тестування, такі як TestRail;

5. Навчання та розвиток команди: ручне тестування вимагає не лише технічних навичок, але й аналітичного мислення та уваги до деталей. Тому, важливо інвестувати в навчання тестувальників, щоб вони могли вдосконалювати свої навички та йти в ногу з останніми тенденціями тестування;

6. Комунікація з розробниками: ефективна комунікація між тестувальниками та розробниками є ключем до швидкого виправлення помилок. Встановлення чітких каналів зв’язку та процесів зворотного зв’язку може значно підвищити продуктивність команди;

7. Розробити контрольний список: контрольний список можна використовувати для перевірки щоденних завдань і переконатися, що під час тестування не пропущені важливі аспекти;

8. Тестування інтерфейсу користувача (UI): Особливу увагу слід приділяти тестуванню інтерфейсу користувача, оскільки воно безпосередньо впливає на досвід користувача. Тестери повинні перевірити не тільки функціональність, але й естетику та інтуїтивність інтерфейсу:

9. Тестування на різних пристроях і платформах: Важливо переконатися, що програмне забезпечення працює належним чином на всіх цільових пристроях і операційних системах, які можуть включати мобільні телефони, планшети, настільні комп’ютери та різні браузери;

10. Використовувати методології Agile та DevOps: інтеграція ручного тестування в практики Agile та DevOps може допомогти забезпечити більшу гнучкість і швидкість виправлення помилок, а також покращити якість кінцевого продукту.

Таким чином, ручне тестування залишається невід’ємною частиною процесу забезпечення якості програмного забезпечення. Воно доповнює автоматизоване тестування, забезпечуючи глибоке розуміння досвіду користувача та виявлення помилок, які інструменти автоматизації можуть пропустити. Чітке планування, розробка тестових випадків, аналіз результатів, вибір відповідних інструментів, навчання та розвиток команди, ефективна комунікація, створення контрольних списків, фокус на UI/UX, тестування на різних пристроях і платформах та інтеграція з Agile та DevOps – це ключ до успішного ручного тестування. Це не тільки покращує якість кінцевого продукту, але й сприяє більш ефективному та гнучкому процесу розробки.

2 РОЗРОБКА РУЧНИХ СЦЕНАРІЇВ ТЕСТУВАННЯ ДЛЯ ВЕБСАЙТІВ

2.1 Вивчення структури та функціональності вебсайту

Перевірка структури та функціональності вебсайту є критично важливим кроком у процесі розробки та тестування, оскільки це закладає основу для забезпечення якості та взаємодії з користувачем.

У сучасному цифровому світі, від інформаційних порталів до електронної комерції, вебсайти стали невід’ємною частиною нашого повсякденного життя. Якість вебсайту безпосередньо впливає на задоволеність користувачів і успіх бізнесу. Таким чином, вивчення структури та функціональності вебсайту є першим і найважливішим кроком у процесі розробки, оскільки це дозволяє розробникам і тестувальникам зрозуміти, як різні елементи вебсайту взаємодіють, і визначити потенційні проблеми, перш ніж вони вплинуть на кінцевого користувача.

Вивчення структури вебсайту допомагає визначити ієрархію та навігацію, які є ключовими для інтуїтивного розуміння для користувачів. Функціональний аналіз, з іншого боку, розкриває можливості для взаємодії з сайтом, такі як форми введення, кнопки та інші інтерактивні елементи. Разом ці аспекти складають основу для створення ефективного інструменту ручного тестування, який гарантує, що вебсайт не тільки добре виглядає, але й функціонує бездоганно. У таблиці 2.1 показано головні структури та функціональності вебсайту.

Таблиця 2.1 – Структура та функціональність вебсайту

Розділ	Опис
Головна сторінка	
Спеціальні пропозиції	Показ акційних продуктів, знижок і спеціальних пропозицій

Продовження таблиці 2.1

Популярні категорії	Вибір найпопулярніших категорій (наприклад, смартфони, планшети, побутова техніка)
Останні новини	Огляд останніх надходжень
Банери	Рекламні банери з актуальними пропозиціями та новинками
Пошук	Поле для швидкого пошуку товарів за назвою або категорією
Категорії	Виділені розділи для популярних категорій
Каталог товарів	
Розширений пошук	Додаткові фільтри для точного вибору товару
Порівняння товарів	Можливість порівнювати характеристики кількох товарів одночасно
Відгуки покупців	Розділ з відгуками, оцінками та рейтингами продуктів
Фільтри	Можливість фільтрувати товари за ціною, брендом, характеристиками
Сортування	Параметри сортування товарів
Картка товару	Зображення, короткий опис, ціна, кнопка «Детально»
Сторінка товару	
Технічні характеристики	Детальний опис параметрів продукту
Відеоогляди	Відео експертів про особливості продукту
Поради та поради	Інформація про найкраще використання продукту
Фотографії	Якісні зображення товару
Опис	Детальний опис товару та його характеристик
Відгуки	Розділ з відгуками клієнтів і рейтингами продуктів
Купити	Кнопка «Додати в кошик» і варіанти доставки

Продовження таблиці 2.1

Кошик	
Зберегти кошик	Можливість зберігати вибрані продукти
Доставка та оплата	Інформація про варіанти доставки та способи оплати
Перегляд	Список вибраних товарів з можливістю змін
Замовлення	Форма для введення інформації про доставку та оплату
Особистий кабінет	
Замовлення	Історія замовлень та їх статус
Історія замовлень	Перегляд попередніх замовлень та їхній статус
Збережені елементи	Можливість додавати елементи до списку збережених
Налаштування	Керування особистими даними та налаштуваннями облікового запису
Підтримка	
FAQ	Розділ з відповідями на поширені запитання
Контактна інформація	Форма зворотного зв’язку та контакти для запитів

У ході цієї бакалаврської роботи детально вивчено та проаналізовано структуру та функціональність вебсайту та важливість різноманітних аспектів у процесі ручного тестування. Виявлено, що глибоке розуміння структури вебсайту має вирішальне значення для ефективного виявлення та усунення помилок і забезпечення високого рівня якості продукту. Функціональність вебсайту, з іншого боку, визначає, як користувачі взаємодіють із вебсайтом, що є ключовим фактором для забезпечення задоволення користувачів.

Аналіз сайту показав, що ретельне тестування кожного аспекту сайту, від навігації до інтерактивних елементів, має вирішальне значення для забезпечення його надійності та зручності використання. Інструкції та ручні методи тестування, розроблені в даній роботі, можна використовувати, щоб зробити процес

тестування більш ефективним, таким чином допомагаючи покращити якість вебсайту.

У майбутньому, у міру розвитку технологій і зміни потреб користувачів, методи й інструменти ручного тестування продовжуватимуть розвиватися. Однак, основні принципи, викладені в цій роботі, залишаються актуальними і слугуватимуть міцною основою для подальших досліджень і розробок у галузі інформаційних технологій.

2.2 Розробка детальних сценаріїв ручного тестування для різних компонентів сайту

Користувачі заходять на сайти з різних пристроїв, браузерів та можуть використовувати різні конфігурації. Щоб забезпечити стабільну роботу вебресурсу за будь-яких умов, потрібно провести сім рівнів тестування:

1. Юніт-тести: суть полягає у перевірці працездатності кожної окремої одиниці програмного коду, щоб функцінувати незалежно від інших функцій. Тестування відноситься до WhiteBox-методів і найчастіше виконується програмістами, а не QA-інженерами;

2. Функціональне тестування сайту: на цьому етапі завдання QA-інженера полягає в тому, щоб перевірити коректність роботи функціоналу майбутнього вебресурсу та його відповідність технічному завданню. Чек лист функціонального тестування сайту включає:

- тестування форм (реєстрації, авторизації та відгук);
- тестування пошуку;
- тестування полів та коректне відображення блоків;
- тестування спливаючих повідомлень;
- тестування фільтрів;
- тестування всіх кнопок;
- перевірка коректного завантаження файлів мультимедіа та їх відправлення на сервер після натискання відповідної кнопки;

- перевірка DevTools: чи всі стилі та зображення завантажуються, чи є помилки в Console;
- перевірка коректної обробки помилок;

3. Інтеграційне тестування: етап QA, під час якого всі окремі функції об’єднуються у групи та тестуються разом як єдиний механізм. У рамках інтеграційного тестування перевіряються: чи працюють сторонні функції належним чином (оплата карткою, соціальні мережі тощо), відображення та аналіз рекламних блоків, показники сайту (кліки, переходи на сторінки, покази блоків вмісту на запитуваних сторінках);

4. Тестування безпеки: забезпечення безпеки даних користувача та захист від кібератак є дуже важливим питанням для будь-якого програмного продукту. Тестування безпеки включає:

- перевірка авторизації: чи може користувач отримати доступ до особистого облікового запису за старим паролем;
- перевірка чи сторінки, які містять важливу інформацію про користувача, мають HTTPS (SSL);
- перевірка стійкісті ресурсу до впровадження SQL і HTML;
- перевірка наявність уразливостей міжсайтового сценарію (XSS);
- перевірка, чи пароль приховано на сторінці авторизації та відновлення облікового запису;
- перевірка ролей та надання доступу до вмісту відповідно до наданих дозволів;

5. Тестування локалізації: перевіряються операції вебсайту в різних країнах з різними форматами дати, числа та валюти. QA-фахівці перевіряють:

- точність відображення дати й часу залежить від часового поясу;
- введення номеру телефону з кодами різних країн;
- якість перекладу при зміні мови сайту;
- коректність позиціонування користувача;
- коректне відображення валют для різних регіонів;

6. Тестування юзабіліті: Usability визначає, наскільки зручно людині користуватись інтернет-ресурсом. Багато в чому цей показник залежить від популярності та продуктивності сайту. У рамках цих тестів перевіряються:

- відсутність помилок у текстах та заголовках;
- доступність підказок для користувачів;
- відсутність непрацюючих посилань та кнопок;
- зручне розміщення контенту;
- корректна робота сайту на різних дозволах екрануК

7. Мультиплатформне тестування: на ринку є десятки браузерів, мільйони моделей пристроїв і вебсайти мають відображатися правильно за будь-яких умов. Щоб досягти цього, виконується мультиплатформне тестування. Воно включає:

- перевірка поведінки вебсайту в популярних браузерах таких як: Opera, Chrome і Edge;
- тестування вебсайту на різних версіях операційних систем;
- перевірка коректної працездатність вебсайту на мобільному пристрої.

У сучасному світі, де користувачі взаємодіють з вебресурсами за допомогою різних пристроїв і браузерів, важливість інтегрованого підходу до тестування незаперечна. Сім рівнів тестування, що охоплює модульне тестування, функціональне тестування, інтеграційне тестування, тестування безпеки, тестування локалізації, тестування зручності використання та крос-платформне тестування, забезпечують комплексну перевірку вебсайту. Це дає змогу виявити й усунути потенційні проблеми, які можуть вплинути на взаємодію з користувачем, безпеку ресурсів і доступність. Цей метод гарантує, що вебсайт може працювати стабільно та ефективно за будь-яких умов, а також забезпечує високу задоволеність користувачів і довіру до мережевих ресурсів.

2.3 Функціональне та користувальницьке тестування: життєвий цикл якісного продукту

Основна мета забезпечення якості — гарантувати, що система програмного забезпечення, розроблена розробниками, відповідає найвищим стандартам і надає клієнтам найкращий продукт або послугу.

Основна мета забезпечення якості (QA) полягає в удосконаленні процесів, щоб надавати продукцію найвищої якості. Для організації вкрай важливо гарантувати оптимальну ефективність і результативність цих процесів.

Щоб гарантувати найвищий рівень якості програмних продуктів, необхідно дотримуватися встановлених стандартів. Щоб досягти бездоганної якості продукції, важливо проводити ретельні оцінки та оцінки.

Життєвий цикл розробки продукту включає як функціональне тестування із залученням розробників, так і ручне тестування із залученням груп потенційних користувачів. На рисунку 1.2 представлено візуальне представлення різних етапів тестування в процесі розробки продукту.

Рисунок 1.2 – Етапи тестування при розробці продукту

Функціональне тестування - це різновид тестування програмного забезпечення, який перевіряє, чи відповідає система функціональним вимогам/специфікаціям. Тестуйте функції, надаючи їм вхідні дані та перевіряючи їх вихідні дані. Функціональне тестування забезпечує правильне виконання вимог. Цей тип тестування пов'язаний не з тим, як відбувається обробка, а з результатами обробки. Він імітує фактичне використання системи, але це не так, ніяких припущень щодо структури системи не робиться.

У процесі функціонального тестування використовується метод Box Testing, тобто, включаючи White Box testing та Black Box testing Рисунок 1.3. White Box testing – це тестування внутрішньої структури, дизайну та кодування. Програмні

рішення, у цьому типі тестування код є видимим для тестувальника. головний фокус. Основна увага приділяється перевірці потоку вхідних і вихідних даних програми та його вдосконаленням для дизайну і простоті використання з додатковою безпекою. White Box testing зазвичай виконується розробниками.

White Box testing передбачає перевірку наступного програмного коду:

1. Питання внутрішньої безпеки;
2. Код пошкоджений або погано побудований;
3. Потік вхідних даних через код;
4. Результати перевірки;
5. Функція умовного циклу;
6. Індивідуальні тести для кожного оператора, об'єкта та функції.

Тестування може проводити систематичне, комплексне та модульне тестування та модульний рівень розробки програмного забезпечення. Одна з головних цілей тестування є whitebox — це перевірка робочого процесу програми він включає тестування послідовності умовних вхідних даних, пов’язаних з очікуваним або бажаним результатом даних.

Black Box testing визначається як техніка тестування, в якій перевірка працездатності тестованої програми, незалежно від внутрішнього знання структури коду, деталей реалізації та внутрішніх шляхів програмного забезпечення. Таке тестування базується виключно на вимогах і специфікаціях програмного забезпечення. У Black-Box Testing розробники зосереджуються на вхідних і вихідних даних програмної системи, не турбуючись про внутрішні процеси цієї програми.

Black-Box може бути будь-якою програмною системою, яку потрібно протестувати. Наприклад, операційні системи, такі як Windows, вебсайти, такі як Google, база даних Oracle. Black-Box Testing дозволяє перевірити ці додатки, воно зосереджується лише на введенні та виведенні, не розуміючи своєї внутрішньої структури коду.

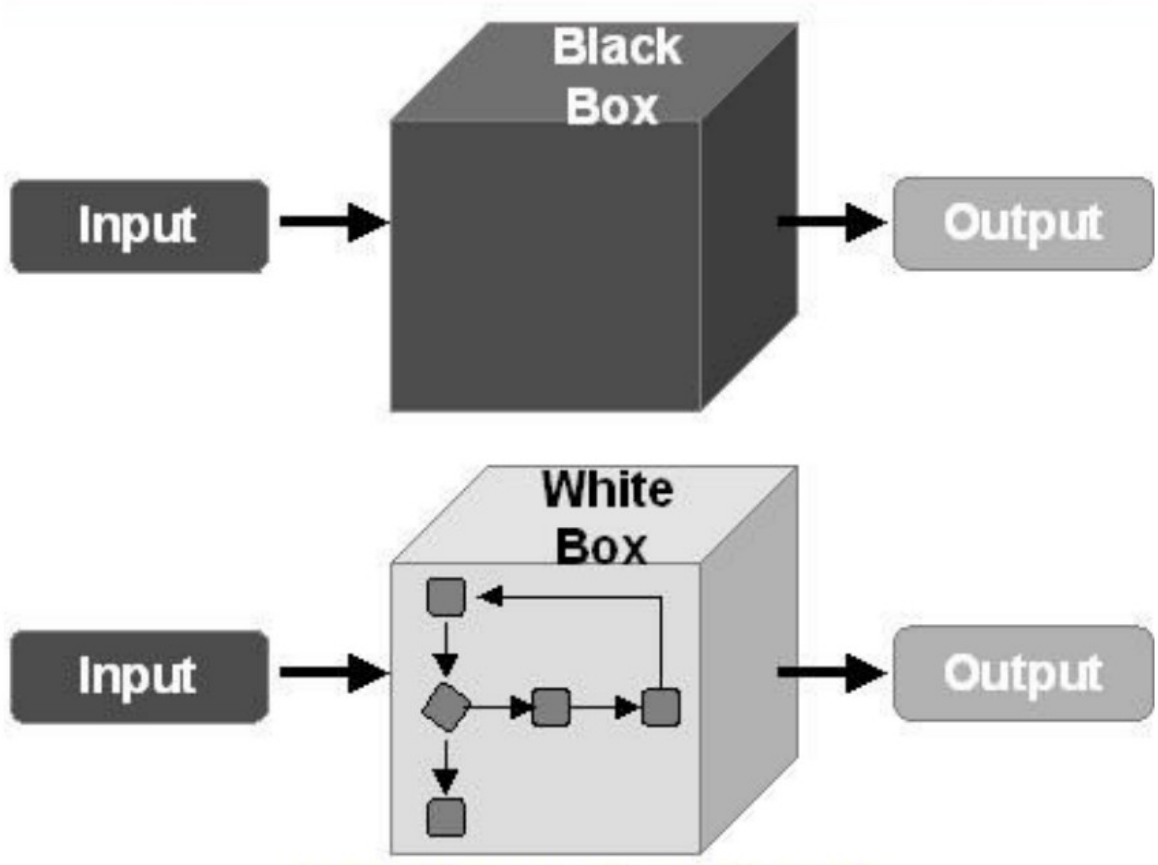


Рисунок 1.3 – Різниця між Black Box Testing та White Box testing

Функціональне тестування зазвичай проводиться на рівнях системного тестування та приймального тестування.

Функціональне тестування складається з наступних етапів:

1. Визначити функції, які має виконувати програмне забезпечення;
2. Створення вхідних даних на основі функціональних характеристик;
3. Визначати висновки на основі характеристики функції;
4. Провести контрольне тестування;
5. Порівняння фактичних результатів з очікуваними.

Функціональне тестування є більш ефективним, коли виконуються умови тестування створюються безпосередньо на основі потреб користувачів або бізнесу. При тестуванні умов створених на основі системної документації (системні вимоги/проєктна документація), дефекти в цьому документації не будуть виявлені під час тестування та можуть стати причиною гніву кінцевих користувачів, коли вони будуть використовувати програмне забезпечення.

Тестування користувача, також відоме як бета-тестування або тестування кінцевим користувачем, що визначається як тестування програмного забезпечення користувачем або клієнтом, щоб визначити, чи можна його прийняти чи ні. Це підсумкові випробування, які проводяться після завершення функціональних випробувань. Основною метою цього тесту є перевірка сумісності програмного забезпечення вимоги до бізнесу. Ця перевірка виконується відомими кінцевими користувачами на вимоги до бізнесу.

Тому що перевірка прийнятності користувачем є останнім тестом, який потрібно виконати перед запуском програмного забезпечення, це, звичайно, остання можливість клієнту протестувати програмне забезпечення та перевірити, чи підходить воно для його цілі.

Кількість етапів, виконаних під час виконання приймального тесту користувача, може змінюватись залежно від того, наскільки деталізовано бажає кожна команда визначити кожен крок процесу. Ці кроки зазвичай включають:

1. Етап планування, під час якого окреслюються бізнес-вимоги та стратегія UAT

2. Ідентифікація та створення тестових сценаріїв. Ці тестові сценарії має охоплювати якомога більше функціональних випадків кінцеві користувачі.

3. Відбираються групи для тестування. Розробники можуть вирішити, щоб кілька кінцевих користувачів тестували програмне забезпечення або відкривали його багатьом іншим користувачам, запропонувавши безкоштовну пробну версію в Інтернеті.

4. Кінцевий користувач починає етап тестування та документування програмного забезпечення, усі можливі помилки та інші проблеми записуються.

5. На етапі виправлення, коли команда розробників модифікує код, виправляє будь-які помилки або вночить запропоновані зміни.

6. Після цих етапів програмне забезпечення готове до випуску у своє виробниче середовище.

Забезпечення якості є важливим аспектом розробки програмного забезпечення, оскільки воно гарантує, що кінцевий продукт відповідає очікуванням клієнтів і стандартам якості. Цей процес включає різноманітні методи тестування, такі як функціональне тестування, White Box та Black Box Testing. Кожен має свої особливості та призначення.

Забезпечення якості є невід’ємною частиною розробки програмного забезпечення та вимагає системного підходу та залучення всіх команд, включаючи розробників, тестувальників і кінцевих користувачів, для досягнення високоякісного кінцевого продукту, який відповідає потребам і очікуванням клієнтів.

2.4 Засоби та технології створення автоматизованих продуктів тестування

Підходів та засобів до тестування продуктів дуже багато, тому перш ніж зробити вибір, потрібно мати уявлення про найбільш використовувані. При цьому потрібно розуміти переваги та недоліки.

Важливо розуміти, що кожен інструмент тестування має різні способи використання: методи, які забезпечують необхідний рівень якості продукції; правильно обрані підходи. Реалізація цієї стратегії забезпечить ефективне використання часу та зусиль, витрачених на тестування. Автоматизація процесів безпосередньо покращує комунікацію та співпрацю між тестувальниками, розробниками та клієнтами, що зрештою призводить до кращого досвіду користувача та більш позитивного сприйняття продукту.

Зараз існує величезне різноманіття мов програмування, які дозволяють проводити тестування продуктів. Найпоширенішими та зручними у використанні є Java, C#, Python, JavaScript та Go.

Java та QA є вирішальною для успішного розроблення програмного забезпечення. Java, як мова програмування, використовується для створення різноманітних додатків, починаючи від вебсайтів і закінчуючи мобільними застосунками. QA відіграє ключову роль у забезпеченні якості цих додатків, перевіряючи їх відповідність вимогам та виявляючи помилки перед тим, як

продукт потрапить до кінцевого користувача. Чому взаємодія між Java та QA є важливою:

1. Забезпечення якості. QA-інженери проводять ретельне тестування, щоб забезпечити відповідність програмного забезпечення встановленим стандартам якості. Вони використовують різні методи тестування, такі як ручне, автоматизоване та тестування безпеки, для виявлення та усунення дефектів. Це сприяє зниженню ризику випуску продукту з помилками, які можуть негативно позначитися на користувацькому досвіді.
2. Автоматизація тестування. Застосування автоматизації тестування дозволяє QA-командам ефективно перевіряти велику кількість сценаріїв та випадків використання продукту. Використання інструментів, таких як Selenium WebDriver, JUnit і інші, сприяє швидкому виявленню помилок та забезпечує більшу точність тестів. Автоматизація також дозволяє проводити тести в нічний час або в неробочі години, що підвищує продуктивність та скорочує час випуску продукту на ринок.
3. Раннє виявлення помилок. Виявлення помилок на ранніх етапах розробки дозволяє командам швидко реагувати та вносити необхідні зміни до коду. Це сприяє зменшенню вартості виправлення помилок та підвищує якість кінцевого продукту. Інтеграція тестування в процес розробки також сприяє кращому розумінню вимог до продукту та його функціональності.
4. Підтримка CI/CD. CI/CD є важливою частиною сучасного процесу розробки, яка дозволяє командам розробників та QA-інженерам працювати разом для швидкого та ефективного випуску оновлень. Автоматизовані тести, написані на Java, інтегруються в CI/CD конвеєри, що дозволяє автоматично тестувати кожну зміну до коду перед її впровадженням у продакшн.
5. Комунікація та співпраця: Ефективна комунікація між розробниками та QA-інженерами є ключовою для успішного процесу розробки. Вона допомагає уникнути непорозумінь та забезпечує, що всі члени команди мають однакове розуміння цілей та очікувань.

Взаємодія Java та QA є необхідною складовою для сучасного процесу розробки програмного забезпечення. Ця співпраця дозволяє створювати високоякісні продукти, які відповідають потребам користувачів та вимогам ринку. Вона сприяє ефективності, надійності та безперервності процесу розробки, а також забезпечує високий рівень задоволеності кінцевих користувачів.

2.5 Системи для застосування тестування в різних мовах програмування

Selenium. Selenium є безкоштовною системою автоматизованого тестування, яка застосовується для перевірки вебдодатків у різних браузерах і на різних платформах. Для створення тестових сценаріїв Selenium можна використовувати кілька мов програмування, таких як Java, C#, Python та інші. Тестування, яке здійснюється за допомогою інструменту тестування Selenium, зазвичай називають "Selenium Testing". Програмне забезпечення Selenium є не лише окремим інструментом, а складається з набору програмних засобів, кожна частина якого задовольняє різні потреби організації в сфері тестування QA.

У складі цього набору знаходяться такі інструменти:

1. WebDriver;
2. Selenium grid;
3. Інтегроване середовище розробки Selenium (IDE);
4. Sel Selenium Remote Control (RC).

WebDriver виявився кращим за Selenium IDE та Selenium RC у багатьох аспектах. У ньому реалізовано більш сучасний і стабільний підхід до автоматизації дій у браузері. На відміну від Selenium RC, WebDriver не покладається на JavaScript для автоматизованого тестування Selenium, а замість цього керує браузером шляхом прямого зв’язку з ним. Повністю підтримується мови, як і в Selenium RC, такі як Java, C#, PHP, Python, та Perl.

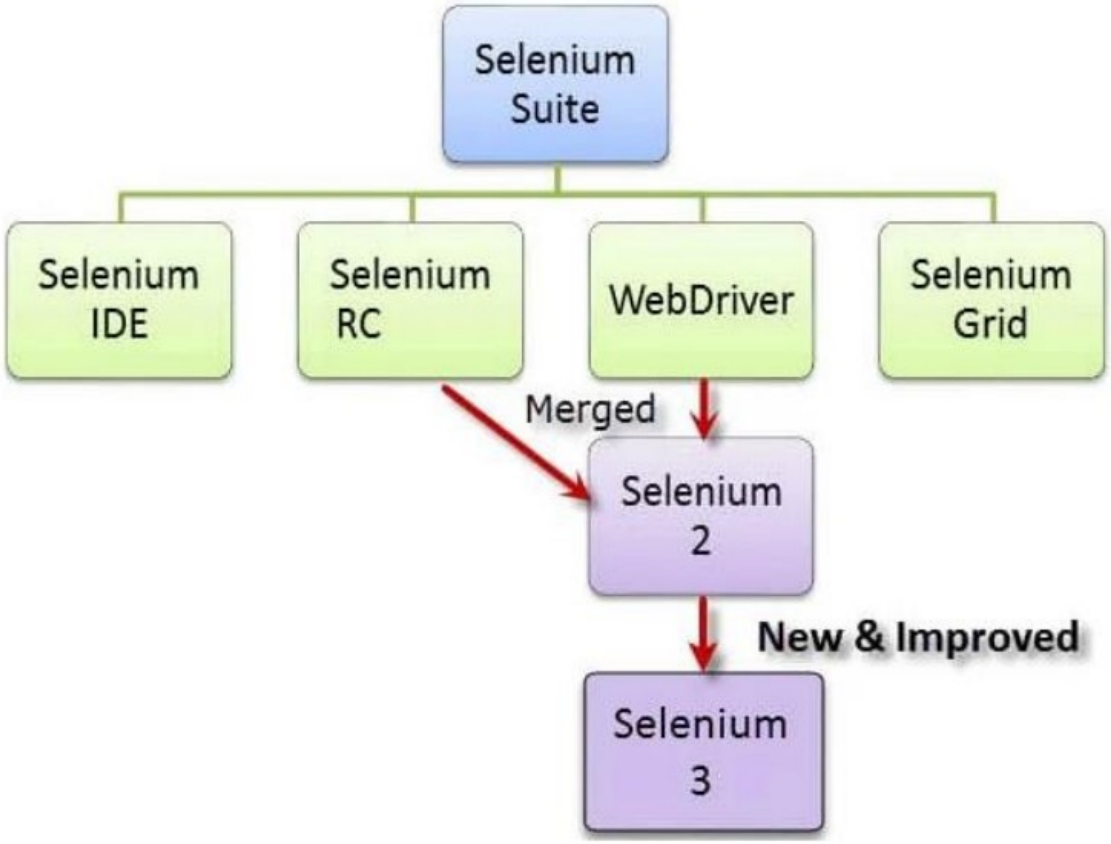


Рисунок 1.4 - Інструменти Selenium

Таблиця 2.2 - Плюси і мінуси Selenium

Переваги	Недоліки
• Простіша установка, ніж у Selenium IDE. • Пряме спілкування з браузером. • Більш реалістична взаємодія з браузером.	• Установка є більш складною. • Вимагає знань з програмування. • Не може широко підтримувати нові браузери. • Немає вбудованого механізму для реєстрації повідомлень про виконання та генерації результатів тестів.

Продовження таблиці 2.2

• Немає необхідності в окремому компоненті, як RC Server. • Швидший час виконання, ніж у IDE та RC.	

Jenkins. Jenkins є сервером безперервної інтеграції з відкритим вихідним кодом, що реалізований мовою програмування Java, з метою упорядкування послідовності дій для досягнення процесу безперервної інтеграції у автоматизованому вигляді. Jenkins забезпечує повний цикл розробки програмного забезпечення, включаючи створення, тестування, документування, розгортання та інші етапи цього циклу. Jenkins широко використовується по всьому світу з близько 300 тисячами встановлень, кількість яких постійно зростає. Завдяки використанню Jenkins компанії-розробники програмного забезпечення можуть значно прискорити процес його розробки, оскільки Jenkins дозволяє швидко здійснювати автоматизацію створення та тестування програмного забезпечення. Цей програмний продукт - серверна програма, що вимагає вебсервер, такий як Apache Tomcat. Популярність програмного забезпечення Jenkins пояснюється його можливістю моніторингу повторюваних завдань, що виникають під час розробки проєкту. Наприклад, у випадку розробки вашою командою проєкту, Jenkins постійно проводитиме тести збірок вашого проєкту та буде показувати помилки на ранніх етапах розробки.

Неперервна інтеграція. Неперервна інтеграція (Continuous Integration) представляє собою процес повторної інтеграції змін у коді, внесених декількома розробниками, в один проєкт. Після фіксації коду програмне забезпечення тестується негайно. Кожен коміт коду призводить до його автоматичного створення та перевірки. У разі успішного проходження тестів, збірка перевіряється для можливості розгортання. При успішному розгортанні код подається до виробництва. Даний процес фіксації, збирання, тестування та

розгортання є безперервним і утворює основу для концепції Continuous Integration/Continuous Deployment (CI/CD). Jenkins - це серверний інструмент, який потребує вебсервера, такого як Apache Tomcat, для функціонування на різних операційних системах, таких як Windows, Linux, macOS, Unix та інші. Для використання Jenkins необхідно створити конвеєри - послідовність кроків, які виконує сервер Jenkins. Конвеєр Continuous Integration Jenkins є потужним інструментом, який складається з низки інструментів, спрямованих на розміщення, контроль, компіляцію та тестування коду або його змін, включаючи такі компоненти як:

- Сервер Continuous Integration (Jenkins, Bamboo, CruiseControl, TeamCity тощо);
- Інструменти управління версіями (наприклад, CVS, SVN, GIT, Mercurial, Perforce, ClearCase тощо);
- Інструменти збирання (Make, ANT, Maven, Ivy, Gradle тощо);
- Платформа для автоматизованого тестування (Selenium, Appium, TestComplete, UFT тощо).

Після того, як розробник зафіксував код, програмне забезпечення було створено та протестовано. У випадку виявлення помилок, відповідний розробник може оперативно виправити їх.

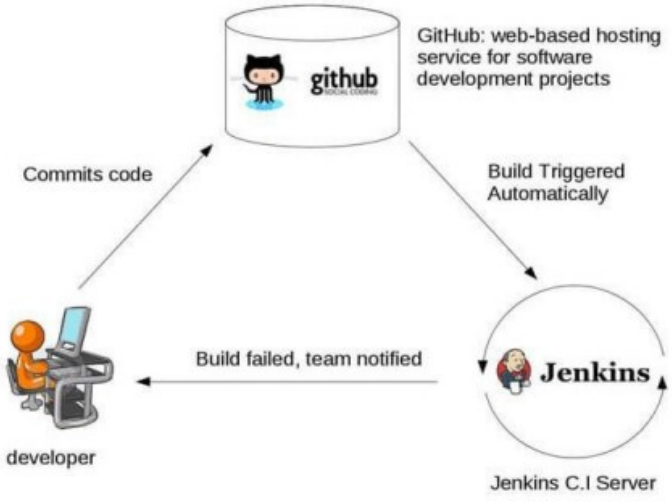


Рисунок 1.5 – Візуальне зображення Jenkins

Переваги використання Jenkins:

1. Зараз керівництво спільнотою Jenkins є відмінним показником його відкритості. Місячні публічні зустрічі та активна участь у процесі розвитку проєкту Jenkins свідчать про це;
2. На даний момент кількість закритих тикетів становить близько 280, і проєкт регулярно випускає стабільні версії кожні три місяці; З прогресом технологій постійно зростає й потужність Jenkins, що підтверджується наявністю близько 320 плагінів у його базі даних. Додавання плагінів робить Jenkins ще більш функціональним та різноманітним;
3. Інструмент Jenkins підтримує хмарну архітектуру, що дає можливість розгортати його на хмарних платформах;
4. Однією з причин, чому Jenkins став такою популярною платформою, є те, що він був розроблений саме для розробників.

Недоліки використання Jenkins:

1. Його інтерфейс вважається застарілим та не зручним порівняно з сучасними тенденціями інтерфейсу користувача;
2. Хоча Jenkins користується популярністю серед розробників, для його підтримки потрібно мати специфічні навички управління сервером, оскільки Jenkins працює на сервері;
3. Складність встановлення та налаштування Jenkins є однією з головних причин, через які багато людей уникають використання цього інструменту;
4. Навіть незначні зміни у налаштуваннях можуть перервати безперервну інтеграцію, що потребує уваги розробника.

Cucumber - це інструмент тестування, який підтримує розвиток, керований поведінкою (BDD). Він пропонує спосіб написання тестів, зрозумілий для всіх, незалежно від їх технічного рівня. У BDD користувачі (бізнес-аналітики, власники продуктів) спочатку пишуть сценарії або приймальні тести, які описують поведінку системи з точки зору клієнта, для перегляду та підписання власниками продукту, перед тим як розробники напишуть свій код. Фреймворк

Cucumber використовує мову програмування Ruby. Припустимо, що вам доручено створити модуль для переказу коштів у програмі Net Banking.

Є кілька способів перевірити його в межах тестування з використанням Cucumber:

- Переказ коштів повинен бути здійснений, якщо на початковому рахунку є достатній баланс;
- Переказ коштів має відбутися, якщо реквізити одержувача правильні;
- Переказ коштів має відбутися, якщо пароль транзакції / код RSA / автентифікація безпеки для транзакції, введені користувачем, правильні; - Переказ коштів має здійснюватися, навіть якщо це вихідний день;
- Переказ коштів має відбутися в дату у майбутньому, встановлену власником рахунку. Сценарій тестування стає більш детальним і складним через додаткові функції, такі як сума переказу X протягом інтервалу Y днів/місяців, зупинка переказу згідно графіку, коли загальна сума досягає Z тощо.

Загальна тенденція розробників полягає в тому, щоб спочатку розробляти функції й лише потім писати тестовий код. Як видно з наведеного вище прикладу, складно розробити тестовий приклад для цього випадку, і розробник відкладе тестування до випуску, після чого проведе швидке, але неефективне тестування. Для подолання цієї проблеми було створено Cucumber BDD (Behavior Driven Development). Це полегшує весь процес тестування для розробника.

Переваги програмного забезпечення Cucumber:

- Вигідно залучати бізнес-зацікавлених сторін, які не можуть легко читати код;
- Інструмент тестування Cucumber зосереджений на досвіді кінцевого користувача;
- Стиль написання тестів дозволяє зручніше використовувати код у тестах;
- Швидке та просте налаштування та виконання;
- Програмне забезпечення Cucumber є ефективним інструментом для тестування.

Rest Assured. Rest Assured дозволяє тестувати REST API за допомогою Java-бібліотек та інтегрується з Maven. Він має дуже ефективні методи порівняння, що робить ствердження очікуваних результатів досить простим. Rest

Assured надає методи для отримання даних з практично будь-якої частини запиту та відповіді, незалежно від складності структури JSON [11]. Для спільноти тестувальників API Automation Testing все ще є новим і нішевим. Складність JSON залишається недослідженою в контексті тестування API, але це не зменшує його важливості у процесі тестування. Фреймворк Rest Assured зробив цей процес дуже простим, використовуючи базові принципи Java, що робить його дуже бажаним для вивчення. Уявіть, що ви відкриваєте карту Google і шукаєте місце, куди хочете піти, і одразу бачите найближчі ресторани, ви бачите варіанти маршруту; від деяких провідних туристичних операторів і маєте на увазі стільки варіантів у вас під рукою. Ми всі знаємо, що вони не продукт Google, тоді як Google вдається це показати. Вони використовують відкритий API цих постачальників. Тепер, якщо вас попросять протестувати такий тип налаштувань, навіть до створення користувацького інтерфейсу або на етапі його розробки, тестування API стає надзвичайно важливим, а їх повторне тестування з різними комбінаціями даних стає дуже практичним у випадку автоматизації. Раніше ми використовували динамічні мови, такі як Groovy, Ruby, для досягнення цього, і це було складно. Тому тестування API не асоціювалося з функціональним тестуванням. Але завдяки Rest Assured, автоматизоване тестування API шляхом надсилання простих HTTP-запитів зручними налаштуваннями стає простим, якщо у вас є базові знання Java. Це необхідно для розуміння тестування API та інтеграційного тестування, але після цього автоматизація, будьте впевнені, заохочує високу впевненість у серверній частині.

Група бібліотек Java під назвою Rest Assured використовується для автоматизації тестування Rest API. Ця бібліотека, що базується на мові програмування Java, дозволяє отримувати значення запиту та відповіді зі складних структур JSON. Крім цього, за допомогою Rest Assured можна налаштовувати запит API шляхом використання різноманітних заголовків, запитів, параметрів шляху та сесій чи файлів cookie. Такий функціонал допомагає встановлювати твердження та умови. Незважаючи на корисність Rest Assured у випадках, коли відповідь має тип JSON, слід зазначити, що його методи можуть

працювати нестабільно у випадках, коли ідентифікатор типу вмісту становить HTML або звичайний текст. Важливо враховувати також ряд корисних порад щодо використання REST-assured, яка є однією з найбільш поширених Java-бібліотек для автоматизації тестування REST-API. Всі наведені приклади базуються на практичному досвіді проведення код-рев'ю в більш ніж 50 проєктах з автотестами. Одним з важливих аспектів є винос end-point'ів в окреме місце для покращення загальної читабельності коду.

У сучасний час кожен виробник та замовник прагне зменшити рівень витрат до мінімуму, одночасно підвищуючи рівень якості кінцевого продукту. Однак не всі особи мають належні знання про те, як досягти цих цілей шляхом використання відповідних підходів та методів тестування і автоматизації в окремих випадках. Отже, оптимальним виходом у цьому контексті буде розроблення належної стратегії тестування та застосування належних інструментів та фахівців. Тестовий автоматизований фреймворк представляє собою набір інструкцій або правил, які використовуються для створення та розробки тестових сценаріїв. Фреймворк складається з комбінації практик та інструментів, що розроблені з метою сприяти більш ефективному тестуванню фахівцями з контролю якості. Ці принципи можуть включати стандарти кодування, методи обробки тестових даних, сховища об’єктів, процеси для збереження результатів тестів або інформацію про отримання доступу до зовнішніх ресурсів. Незважаючи на те, що це не є обов'язковими правилами, і тестувальники все одно можуть складати сценарії або проводити тести, не дотримуючись їх, використання організованої структури як правило надає додаткові переваги, яких в іншому випадку вони б не мали. Тестовий автоматизований фреймворк є зручним інструментом для полегшення рутинної роботи тестувальників. Важливим аспектом є належне створення його структури та написання необхідних тест-сценаріїв. При правильному підході результати тестування та стан "здоров'я" продукту зможе перевірити будь-яка особа, що розуміє специфіку проєкту завдяки зрозумілій системі звітності. Це буде корисним не лише для тестувальників чи керівництва, а й для розробників, які

матимуть змогу перевіряти якість своїх змін завдяки швидкому проведенню модульних тестів після кожного оновлення коду проєкту.

3 СТВОРЕННЯ ФРЕЙМВОРКУ ДЛЯ ТЕСТУВАННЯ

Широке розгортання та участь у життєвому циклі проєкту автоматизації тестування цього фреймворку завжди корисний і актуальний. При збільшенні архітектури, зростаючий розмір і функціональність додатків неможливо реалізувати без їх використання.

Зрештою, важко уявити, скільки для цього знадобиться манулаьних тестувальників, щоб перевірити вебсторінку чи інший сайт з більш високим рівнем популярності і скільки там помилок, вони можуть помилитися в цьому через втому і рутинну роботу.

Загалом, перехід від ручного тестування до автоматизованого - процес є цілком логічним кроком для досягнення високої якості кінцевого продукту. Однак важливо, щоб ця структура була максимально універсальною, безпечною та простою в редагуванні та не залежить від платформи чи браузера. В цьому сенс розробити автоматизовану структуру для тестування, що буде легко працювати над своїм проєктом на всіх платформах і сприяти його вдосконаленню і одержувати найвищу якість.

3.1 Створення фреймворку за допомогою Java

Наша структура базуватиметься на загальній програмі
на мові програмування Java, яку ми розроблятимемо з додатками та налаштуваннями.
Однак при створенні ми будемо використовувати більш продвинуті налаштування та створити проєкт за допомогою
ресурсів maven.

Maven — це засіб автоматизації роботи з програмними проєктами, який спочатку використовувався для Java проєктів. Використовується для керування та складання програм. Maven забезпечує підтримку побудови не просто перебираючи файли з цього репозиторію, але й завантажуючи назад артефакти у кінці побудови. Локальний кеш звантажених артефактів діє як первісний засіб синхронізації виходу проєктів на локальній системі.

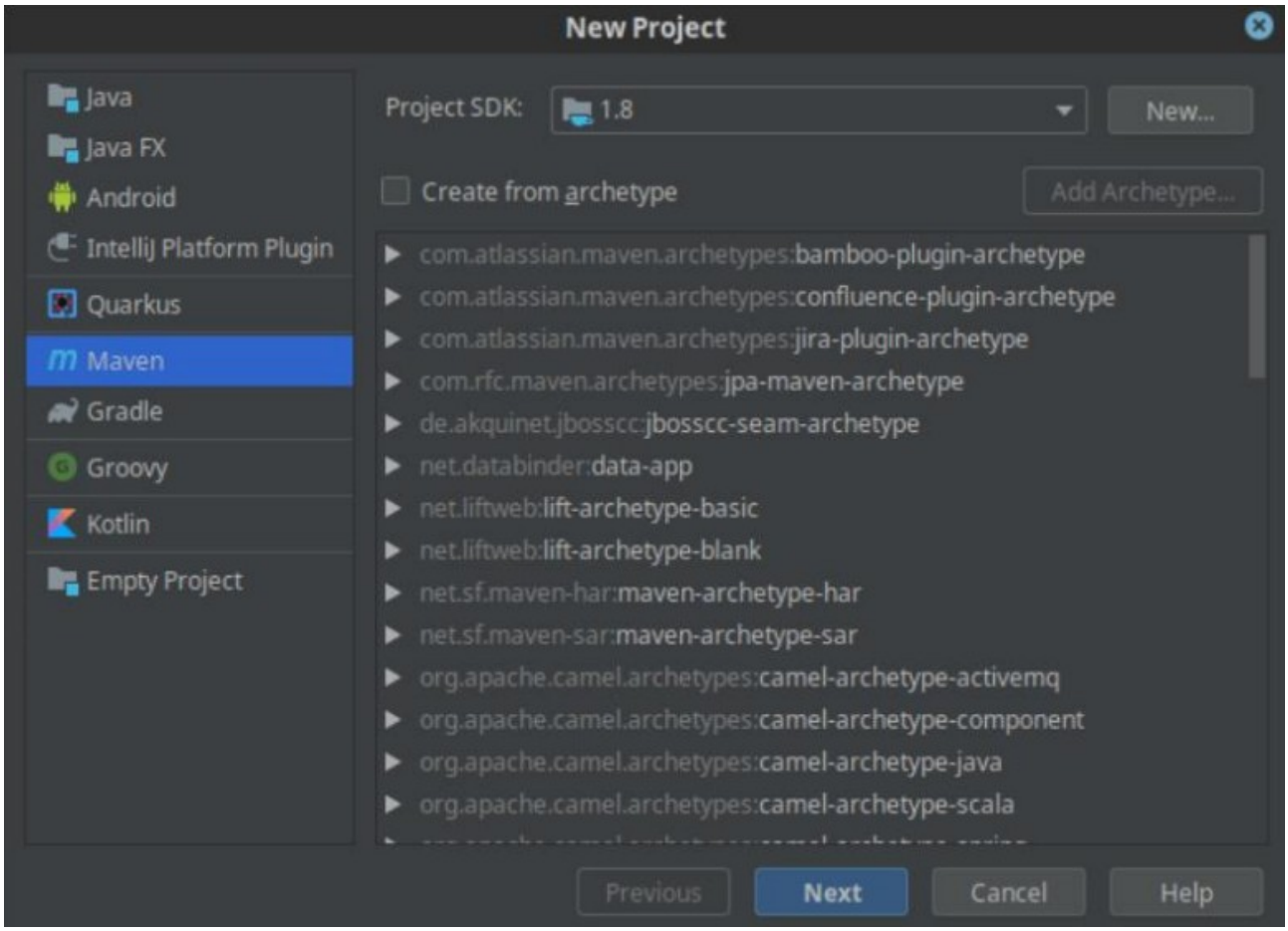


Рисунок 3.1 – Вигляд конфігурації Maven в Intellij

При створенні такого проєкту ми можемо доповнювати його. У певному середовищі розробки підтримуються різні налаштування і на певній мові програмування, шляхом створення pom. файлів. Додавши все необхідне і завершивши реалізацію проєкту, ми можемо продовжувати писати класи в нашій структурі.

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <project xmlns="http://maven.apache.org/POM/4.0.0"
3            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4            xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5       <modelVersion>4.0.0</modelVersion>
6
7       <groupId>onfreetube</groupId>
8       <artifactId>ideaProjectOnfreeTube</artifactId>
9       <version>1.0-SNAPSHOT</version>
10
11
12  </project>
```

Рисунок 3.2 – Приклад pom файлу

Після додавання всього необхідного та завершення налаштування проєкту ми можемо продовжити писати класи безпосередньо в нашій структурі.

3.2 Створення тестових сценаріїв для частини інтерфейсу користувача

Давайте розглянемо один із найважливіших класів драйверів, дозволи нам створити сутності браузера для тестування інтерфейсу користувача.

Його реалізація наведена нижче:

```
public class Driver {
    private static final Logger LOG = LoggerFactory.getLogger(Driver.class);

    private static final String WORKING_DIR = System.getProperty("user.dir");

    private static final String PATH_TO_DOWNLOADS = WORKING_DIR +
File.separator + "downloads";

    private static String browserType =

Optional.ofNullable(TestContext.getInstance().getBrowserType()).orElse(Browse
rType.CHROME).toLowerCase();

    private static ThreadLocal<WebDriver> DRIVER = new
ThreadLocal<WebDriver>();

    private static List<WebDriver> storedDrivers = new ArrayList<>();

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

    public static void removeDriver() {
        if (DRIVER.get() != null) {
            storedDrivers.remove(DRIVER.get());
            DRIVER.get().quit();
            DRIVER.remove();
        }
    }

    public static void addDriver() {
        String platformRunType =
Optional.ofNullable(System.getProperty("platform")).orElse("local").toLowerCa
se();
        WebDriver createdDriver = platformRunType.equals("remote") ?
getSelenoidDriver() : getLocalDriver();
        storedDrivers.add(createdDriver);
        DRIVER.set(createdDriver);
    }
```

```
    public static String getBrowserType() {
        return browserType;
    }

    static {
        Runtime.getRuntime().addShutdownHook(new Thread(() ->
storedDrivers.stream().forEach(WebDriver::quit)));
    }

    //
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////
    // Driver creation
    //
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////

    public Driver() {
    }

    private static WebDriver getLocalDriver() {
        WebDriver builtDriver;
        switch (browserType) {
            case BrowserType.CHROME:
                builtDriver = buildChrome();
                break;
            case BrowserType.IE:
                builtDriver = buildIE();
                break;
            case BrowserType.SAFARI:
                builtDriver = buildSafari();
                break;
            default:
                browserType = BrowserType.CHROME;
                builtDriver = buildChrome();
        }
        builtDriver.manage().window().maximize();
        return builtDriver;
    }
```

```
        ChromeOptions options = new ChromeOptions();
        Map<String, Object> prefs = new HashMap<>();

        //turn off message "Let's save your password for this site"
        prefs.put("credentials_enable_service", false);
        prefs.put("profile.password_manager_enabled", false);

prefs.put("profile.content_settings.exceptions.automatic_downloads.*.setting"
, 1);

        //
        prefs.put("download.default_directory", PATH_TO_DOWNLOADS);
        options.setExperimentalOption("prefs", prefs);

        //turn off message "Chrome is being controlled by automated test
software"
        options.setExperimentalOption("excludeSwitches",
Collections.singletonList("enable-automation"));
        options.setExperimentalOption("useAutomationExtension", false);
        //
        options.addArguments("--no-sandbox");
        options.addArguments("--disable-logging");
        options.addArguments("--start-maximized");
        options.addArguments("--disable-extensions");
        options.addArguments("--disable-web-security");
        options.addArguments("--disable-notifications");
        options.addArguments("--no-default-browser-check");

        options.setCapability(CapabilityType.HAS_NATIVE_EVENTS, true);
        options.setCapability(CapabilityType.SUPPORTS_JAVASCRIPT, true);
        return options;
    }
```

```
    private static WebDriver buildIE() {
        Capabilities capabilities = getIEDesiredCapabilities();
        WebDriverManager.iedriver().arch32().setup();
        return new ThomsonReutersInternetExplorerDriver(capabilities);
    }

    private static DesiredCapabilities getIEDesiredCapabilities() {
        DesiredCapabilities capabilities =
DesiredCapabilities.internetExplorer();
        capabilities.setCapability("requireWindowFocus", false);
        capabilities.setCapability(InternetExplorerDriver.NATIVE_EVENTS,
true);

capabilities.setCapability(InternetExplorerDriver.INTRODUCE_FLAKINESS_BY_IGNO
RING_SECURITY_DOMAINS, true);

capabilities.setCapability(InternetExplorerDriver.IE ENSURE CLEAN SESSION,
true);

capabilities.setCapability(InternetExplorerDriver.ENABLE_PERSISTENT_HOVERING,
true);

capabilities.setCapability(InternetExplorerDriver.UNEXPECTED_ALERT_BEHAVIOR,
"accept");
```

```
capabilities.setCapability(InternetExplorerDriver.IGNORE_ZOOM_SETTING, true);

capabilities.setCapability(CapabilityType.ForSeleniumServer.ENSURING_CLEAN_SE
SSION, true);
        capabilities.setCapability("unhandledPromptBehavior", "dismiss");
        capabilities.setCapability("platformName", "windows");
        capabilities.setCapability("ignoreProtectedModeSettings", true);
        capabilities.setCapability("disable-popup-blocking", true);
        capabilities.setJavascriptEnabled(true);
        return capabilities;
    }
```

```
    private static WebDriver buildSafari() {
        LOG.info("Initializing Safari driver...");
        SafariOptions options = new SafariOptions();
        options.setCapability("browserName", "safari");
        return new ThomsonReutersSafariDriver(options);
    }
}
```

Використовуючи цей клас, ми можемо створити кілька сутностей найпопулярніших браузерів для наших тестів GUI user. Тепер давайте подивимося на базовий клас у нашій структурі і від нього всі далі будуть успадковувати.

```
public class BasePage {

    private static final Logger LOG =
LoggerFactory.getLogger(BasePage.class);

    public static final String DOWNLOAD_PATH = getProperty("user.dir") +
separator + "downloads" + separator;

    public static String savedId;

    protected static final String GRID_STATUS = ".dgrid-status";

    protected static final String GLOBE_ICON_IN_LIST = ".icon-globe";

    protected static final String SUCCESS_MESSAGE = ".messageContent";

    protected static final String LOADING_INDICATOR = ".loadingSpinner";

    protected static final String DIALOG_UNDERLAY = ".dijitDialogUnderlay";

    private static final int DEFAULT_POLLING_IN_MILLS = 500;

    private static final int AFTER_SCENARIO_CLEANUP_WAIT = 2000;
```

```
    private static final int AFTER_SCENARIO_CLEANUP_ATTEMPTS = 60;

    private static final int AFTER_SCENARIO_CLEANUP_DB_CHECK_ATTEMPTS = 2;

    private static final String SHOWN = "shown";

    private static final String HIDDEN = "hidden";

    private static final String LOGO = ".logo";

    private static final String TEST = "test ";

    private static final String ASSIGNED_USER = "bdd testuser";

    private static final String PROXY_HOST_PROPERTY = "https.proxyHost";

    private static final String PROXY_PORT_PROPERTY = "https.proxyPort";

    private static final String ERROR_STATUS_MESSAGE =
            "Incorrect status parameter '%s', correct are: 'shown' or
'hidden'";

    private static final String VALIDATION_MESSAGE_ERROR =
"div.wrapper.error";

    private static final String CUSTOM_FIELD = "TESTCF" +
UUID.randomUUID().toString();

    private static final String VALIDATION_MESSAGE =
"div.wrapper.error,div.wrapper.success";

    private static final String STATUS_MESSAGE_LOCATOR = ".statusMessage
.messageContent";

    private static final By STATUS_MESSAGE_CSS =
By.cssSelector(".statusMessage .messageContent");

    private static final By MAIN_MENU_ITEMS =
            new By.ByXPath("//coral-tab[@level='3']//span");

    private static final String RESET_LOADER = ".CaseManager
.loadingIndicatorView:not(.hidden)";

    private static final String DIALOG_MESSAGE_RIGHT = ".dijitTooltipRight
.dijitTooltipContents";
```

```
    private static final String DIALOG_MESSAGE_BELOW =
".dijitTooltipDialogPopup .dijitTooltipContents";

    private static final String RUN_TIME_FILES_PATH = getProperty("user.dir")
+ separator + "testFiles" + separator;

    private static final String HOVER_OVER_ELEMENT_JS = "var evObj =
document.createEvent('MouseEvents');"
            + "evObj.initMouseEvent(\"mouseover\",true, false, window, 0, 0,
0, 0, 0, false, false, false, false, 0, null);"
            + "arguments[0].dispatchEvent(evObj);";

    private static final String CANCEL_HOVER_OVER_ELEMENT_JS = "var evObj =
document.createEvent('MouseEvents');"
            + "evObj.initMouseEvent(\"mouseout\",true, false, window, 0, 0,
0, 0, 0, false, false, false, false, 0, null);"
            + "arguments[0].dispatchEvent(evObj);";
```

```
private static Map<String, String> testContext = new HashMap<>();

private static Map<Object, Object> testSession = new HashMap<>();

private static Map<String, String> groupIdReferenceMap = new HashMap<>();

public Waiters waiters = new Waiters(Driver.getDriver());

protected Actions action = new Actions(Driver.getDriver());

private SetUpRest setupRest = new SetUpRest();

private String host = getProperty("host");

@FindBy(xpath = "//link[@rel='shortcut icon']")
private WebElement favicon;

@FindBy(css = "div.world-check-one-logo")
private WebElement navbarLogo;

@FindBy(css = ".statusMessage .messageContent")
private WebElement statusMessage;

@FindBy(css = ".wrapper span.close")
private WebElement closePersistenceMessageIcon;

@FindBy(xpath = "//coral-tab[@active='true']//span")
private WebElement activePageText;

@FindBy(css = ".windowApp")
private WebElement frame;

@FindBy(css = ".dijitAlignLeft .content.parent .name")
private WebElement parentGroup;

@FindBy(css = "a.whatsNew")
private WebElement whatsNewLink;

@FindBy(css = ".home-button")
private WebElement homeIcon;
```

```
    public BasePage() {
        PageFactory.initElements(new ProjectFieldDecorator(new
DefaultElementLocatorFactory(Driver.getDriver())), this);
    }

    public String getPageTitle() {
        return Driver.getDriver().getTitle();
    }

    public Actions getAction() {
        return action;
    }

    //
////////////////////////////////////////////////////////////////////////////////
////////////////////////////
    // Element
    //
////////////////////////////////////////////////////////////////////////////////
////////////////////////////
    public WebElement findElement(By by) {
        return Driver.getDriver().findElement(by);
```

```
    }


    public List<WebElement> findElements(By by) {
        return Driver.getDriver().findElements(by);
    }

    //
////////////////////////////////////////////////////////////////////////////////
////////////////////////////
    // Element status
    //
////////////////////////////////////////////////////////////////////////////////
////////////////////////////
    public boolean isElementPresent(By locator) {
        try {
            return findElement(locator) != null;
        } catch (NoSuchElementException e) {
            return false;
        }
    }

    public boolean isElementPresent(WebElement element) {
        try {
            return element.getLocation() != null;
        } catch (NullPointerException | NoSuchElementException |
StaleElementReferenceException e) {
            return false;
        }
    }

    public boolean isElementDisplay(WebElement webElement) {
        try {
            return webElement.isDisplayed();
        } catch (NoSuchElementException | StaleElementReferenceException e) {
            return false;
        }
    }

    public boolean isElementClickable(WebElement webElement) {
        try {
            return webElement.isDisplayed() && webElement.isEnabled();
        } catch (NoSuchElementException | StaleElementReferenceException e) {
            return false;
        }
    }
```

```
    }

    public boolean isElementClickable(By locator) {
        try {
            return isElementClickable(findElement(locator));
        } catch (NoSuchElementException | StaleElementReferenceException e) {
            return false;
        }
    }

    public boolean isElementDisplay(By locator) {
        try {
            return findElement(locator).isDisplayed();
        } catch (NoSuchElementException e) {
            return false;
        }
    }

    public boolean isNestedElementDisplayed(WebElement parent, By by) {
```

```
        try {
            if (isElementPresent(parent.findElement(by))) {
                scrollIntoView(parent.findElement(by));
            }
            return isElementDisplay(parent.findElement(by));
        } catch (NoSuchElementException | StaleElementReferenceException e) {
            return false;
        }
    }

    public void verifyDisplayStatusOfElement(String displayStatus, By
locator) {
        switch (displayStatus) {
            case SHOWN:
                waiters.waitForElementToBeDisplay(locator);
                break;
            case HIDDEN:
                waiters.waitForElementToDisappear(locator);
                break;
            default:
                throw new
IllegalArgumentException(String.format(ERROR STATUS MESSAGE, displayStatus));
        }
    }

    public void verifyDisplayStatusOfElement(String displayStatus, WebElement
webElement) {
        switch (displayStatus) {
            case SHOWN:
                waiters.waitForElementToBeDisplay(webElement);
                break;
            case HIDDEN:
                waiters.waitForElementToDisappear(webElement);
                break;
            default:
                throw new
IllegalArgumentException(String.format(ERROR_STATUS_MESSAGE, displayStatus));
        }
    }
```

```
            LOG.error("Element isn't user editable");
        }
        Driver.getDriver().findElement(by).click();
        Driver.getDriver().findElement(by).sendKeys(dataToSend);
    }

    public void inputText(WebElement element, CharSequence... dataToSend) {
        waiters.waitForElementToBeDisplay(element);
        element.clear();
        element.sendKeys(dataToSend);
    }

    public void inputTextNoClean(WebElement element, CharSequence...
dataToSend) {
        waiters.waitForElementToBeDisplay(element);
        element.sendKeys(dataToSend);
    }

    public void clickOnElement(WebElement element) {
        try {
            waiters.waitForPresenceOfElement(element);
            waiters.waitForElementToBeClickable(element).click();
        } catch (WebDriverException e) {
            throw new IllegalStateException("WebDriver exception encountered:
" + e.getMessage(), e);
        }
    }

    public void clickOnElementWithDelay(By locator) {
        try {
            waiters.waitForElementToBeClickable(locator).click();
        } catch (WebDriverException e) {
            throw new IllegalStateException(
                    "Element found by locator '" + locator + "' not found on
the page: " + e.getMessage(), e);
        }
    }

    public void clickOnElement(By locator) {
        try {
            waiters.waitForElementToBeClickable(locator).click();
        } catch (WebDriverException e) {
            throw new IllegalStateException(
                    "Element found by locator '" + locator + "' not found on
the page: " + e.getMessage(), e);
        }
```

```
        }
    }

    public void moveToElementUsingJS(WebElement element) {
        ((JavascriptExecutor)
Driver.getDriver()).executeScript(HOVER_OVER_ELEMENT_JS, element);
    }

    public void cancelMoveToElementUsingJS(WebElement element) {
        ((JavascriptExecutor)
Driver.getDriver()).executeScript(CANCEL_HOVER_OVER_ELEMENT_JS, element);
    }

    public void actionMoveToElement(WebElement element) {
        getAction().moveToElement(element).perform();
    }

    public void actionClickOnWebElement(WebElement element) {
        waiters.waitForElementToBeClickable(element);
```

```
    public void enterText(WebElement element, CharSequence... dataToSend) {
        waiters.waitForElementToBeDisplay(element);
        waiters.waitForElementToBeClickable(element);
        element.clear();
        element.click();
        element.sendKeys(dataToSend);
    }

    // Workaround for stale element fix
    public void enterText(By by, CharSequence... dataToSend) {

waiters.waitForElementToBeDisplay(Driver.getDriver().findElement(by));

waiters.waitForElementToBeClickable(Driver.getDriver().findElement(by));
        try {
            Driver.getDriver().findElement(by).clear();
        } catch (InvalidElementStateException e) {
```

```
        getAction().moveToElement(element).click(element).build().perform();
    }

    public void minimizeBrowserWindow() {
        Driver.getDriver().manage().window().setSize(new Dimension(300,
500));
    }

    public void maximizeBrowserWindow(){
        Driver.getDriver().manage().window().maximize();
    }

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

    public void switchToFrame() {
        waiters.waitForElementToBeDisplay(frame);
        Driver.getDriver().switchTo().frame(frame);
    }

    public void switchToFrameIfExists() {
        if (isElementDisplay(frame)) {
Driver.getDriver().switchTo().frame(frame);}
    }

    public void refreshPage() {
        LOG.info("Refreshing page...");
        boolean isCurrentFrameFspApp =

waiters.waitForPresenceOfElement(By.tagName(BODY.getValue())).getAttribute(CL
ASS.getAttributeValue())
                        .contains(FSP_APP);
        Driver.getDriver().navigate().refresh();
        if (isCurrentFrameFspApp) {
            switchToFrame();
        }
    }
```

```
    /**
     * Transforms List of WebElements to List of Strings
     *
     * @param elements list of WebElements
     * @return transformed List of Strings
     */
    public List<String> transformElementToText(List<WebElement> elements) {
        return
elements.stream().map(WebElement::getText).collect(Collectors.toList());
    }

    /**
     * Verify equality of transformed WebElement List with List of Strings
     *
     * @param stringList:     list of Strings
     * @param webElementList: list of WebElements
     * @return true if lists content equals
```

```
     */
    public boolean isWebElementListEqualsToStringList(List<String>
stringList, List<WebElement> webElementList) {
        return
stringList.equals(transformElementToText(waiters.waitForAllElementsToBeDispla
y(webElementList)));
    }

    /**
     * @param webElementList: list of WebElements
     * @param elements:       list of Strings
     * @return true if WebElements list doesn't contain elements from list of
Strings
     */
    public boolean isWebElementListNotContainsElements(List<WebElement>
webElementList, List<String> elements) {
        return webElementList.stream()
                .noneMatch(item -> elements.contains(item.getText()));
    }

    /**
     * Asserts order of list of WebElements
     *
     * @param expected list
     * @param elements list of WebElements
     */
    public void assertListOrder(List<String> expected, List<WebElement>
elements) {
        List<String> elementsAsText = transformElementToText(elements);
        assertThat(elementsAsText, contains(expected.toArray()));
    }

    /**
     * Asserts list of WebElements in any order
     *
     * @param expected list
     * @param elements list of WebElements
     */
    public void assertListAnyOrder(List<String> expected, List<WebElement>
elements) {
        List<String> elementsAsText = transformElementToText(elements);
        assertThat(elementsAsText, containsInAnyOrder(expected.toArray()));
    }
```

```
    /**
     * Asserts expected and actual text in web page
     *
     * @param element      String element
     * @param expectedText String text
     */
    public void assertTextOnWebElement(WebElement element, String
expectedText) {
        String actualText =
waiters.waitForElementToBeDisplay(element).getText();
        assertThat(actualText, containsString(expectedText));
    }

    /**
     * Asserts WebElement is not displayed
     *
     * @param cssSelector String used inside css selector
     */
    public void assertWebElementNotPresent(String cssSelector) {
        assertThat("Error: Element is present by cssSelector " + cssSelector,
```

```
    public List<String> getListedMenuItems() {
        switchToDefaultContent();
        List<WebElement> elements =
waiters.waitForAllElementsToBeDisplay(MAIN_MENU_ITEMS);
        return elements.stream()
                .map(WebElement::getText)
                .collect(Collectors.toList());
    }

    public boolean isStatusMessageContains(String expectedMessage) {
        String statusMessage = getValidationMessage().toLowerCase();
        waitForInvisibilityOfValidationMessage();
        if (!StringUtils.containsIgnoreCase(statusMessage, expectedMessage))
{
            LOG.error(String.format("Actual: '%s'. Expected: '%s'.",
statusMessage, expectedMessage));
            return false;
        }
        return true;
    }

    public void assertTextOnNavActivePage(String text) {
        switchToDefaultContent();
        assertTextOnWebElement(activePageText, text);
        switchToFrame();
    }

    public String getValidationMessage() {
        try {
            return
waiters.waitForElementToBeDisplay(statusMessage).getText();
        } catch (TimeoutException e) {
            LOG.error("Notification Message for Action was not displayed.
Getting text from hidden element.", e);
            try {
                return getTextByJavascript(statusMessage);
            } catch (NoSuchElementException e2) {
                throw new RuntimeException("Notification Status Message is
not present", e2);
            }
        }
    }
```

```
    protected String getTextByJavascript(final WebElement element) {
        String script = "var element = arguments[0];" + "return
element.textContent;";
        return (String) ((JavascriptExecutor)
Driver.getDriver()).executeScript(script, element);
    }

    public String getValidationMessageColor() {
        try {
```

```
waiters.getNewWait(5).until(ExpectedConditions.visibilityOfElementLocated(STA
TUS_MESSAGE_CSS));
        } catch (TimeoutException e) {
            return "no banner";
        }
        return
findElement(By.cssSelector(VALIDATION_MESSAGE_ERROR)).getCssValue(BACKGROUND_
COLOR.getValue());
    }

    public void waitForValidationMessageDisappear() {

waiters.waitForElementToDisappear(By.cssSelector(VALIDATION_MESSAGE));
    }

    public void waitForInvisibilityOfValidationMessage() {
        waiters.waitForElementToDisappear(STATUS_MESSAGE_LOCATOR);
    }

    public void waitForVisibilityOfValidationMessage() {
        waiters.waitForElementToBeDisplay(STATUS_MESSAGE_LOCATOR);
    }

    public String getAlertDialogMessageBelow() {
        return
waiters.waitForElementToBeDisplay(By.cssSelector(DIALOG_MESSAGE_BELOW)).getTe
xt();
    }

    public String getAlertDialogMessageRight() {
        return
waiters.waitForNonBlankElementText(By.cssSelector(DIALOG_MESSAGE_RIGHT));
    }
    }
```

Успадкувавши цей клас, ми можемо створювати наші класи уособлення сторінок нашої вебсторінки. Наведено приклад такого класу нижче:

```
public class Login  extends BasePage {

    private static final Logger LOG = LoggerFactory.getLogger(Login.class);

    private static final SubscriptionPage subscriptionPage =
PageFactory.initElements(Driver.getDriver(), SubscriptionPage.class);

    public static TestContext testContext = TestContext.getInstance();

    public static String user = testContext.getUser();

    public static String password = testContext.getPassword();

    private String SIGN_OUT_LOCATOR = "//*[contains(text(), '%s')]";

    @FindBy(css = "input[type='text']")
    private WebElement userNameSelector;

    @FindBy(css = "input[type='password']")
    private WebElement passwordSelector;

    @FindBy(css = ".button_img")
    private WebElement signInButton;

    @FindBy(css = ".button_75")
    private WebElement continueSignInCss;

    @FindBy(css = ".body_message a")
    private WebElement signBackInLink;

    @FindBy(xpath ="//*[contains(@data-bind, 'SkipMfaRegistration')]")
    private WebElement skipRegistration;

    @FindBy(css = "[type='Submit']")
    private WebElement nextButton;

    @FindBy(xpath ="//input[@name='IDToken2' and @type='password']")
    private WebElement choosePassword;

    @FindBy(xpath ="//input[@name='IDToken3' and @type='password']")
    private WebElement confirmPassword;

    @FindBy(xpath ="//div[text()='Next']")
    private WebElement setPasswordNextButton;

    public void setPassword(String password) {
        enterText(choosePassword, password);
        enterText(confirmPassword, password);
        clickOnElement(setPasswordNextButton);
    }

    public void login() {
        login(user, password);
    }

    public void login(String user, String password) {
            LOG.info("Navigate to: " + Url.url());
            Driver.getDriver().navigate().to(Url.url());
            LOG.info("Logging as   user...");
            enterText(userNameSelector, user);
            enterText(passwordSelector, password);
            signInButton.submit();
            continueSignInIfLoggedInElsewhere();
    }
```

```
    public void eikonLogin() {
        By usernameInput = By.cssSelector("input[type='text']");
        By passwordInput = By.cssSelector("input[type='password']");
        By signInButton = By.xpath("//div[text()='Sign In']");
        enterText(usernameInput, user);
        enterText(passwordInput, password);
        clickOnElement(signInButton);

        if (Driver.getDriver().getTitle().equalsIgnoreCase("You are signed in
to another device")) {
            clickOnElement(signInButton);
        }
    }

    public void loginSpecifyingRegKey(String user, String password, String
userRegKeyLabel) {
        try {
            login(user, password);
        } catch (TimeoutException e) {
            LOG.info("No login form, as we are logged in before.");
        }
        subscriptionPage.selectUserRegKeyIfRequired(userRegKeyLabel);
    }

    public void loginBookmarkSpecifyingRegKeyLabel(String user, String
password, String userRegKeyLabel) {
        LOG.info("Navigate to: " +
TestContext.getInstance().getProperty("BOOKMARK_LOGIN_URL"));

Driver.getDriver().navigate().to(TestContext.getInstance().getProperty("BOOKM
ARK_LOGIN_URL"));
        LOG.info("Bookmarked Logging as   user...");

        enterText(userNameSelector, user);
        enterText(passwordSelector, password);
        signInButton.submit();
        if (isElementDisplay(signInWhenUserAlreadySignedIn)) {
            LOG.info("SignIn click.");
            clickOnElement(signInWhenUserAlreadySignedIn);
        }
        subscriptionPage.selectUserRegKeyIfRequired(userRegKeyLabel);
    }

    public void loginSsoSpecifyingRegKeyLabel(String user, String password,
String userRegKeyLabel) {
        LOG.info("Navigate to: " +
TestContext.getInstance().getProperty("SSO_LOGIN_URL"));

Driver.getDriver().navigate().to(TestContext.getInstance().getProperty("SSO_L
OGIN_URL"));
        LOG.info("Logging via SSO...");

        enterText(ssoLogin, user);
        clickOnElement(nextButton);

        enterText(ssoPassword, password);
        clickOnElement(nextButton);

        if (isElementDisplay(skipRegistration)) {
            LOG.info("Skip registration click.");
            clickOnElement(skipRegistration);
        }
```

```
        }
        clickOnElement(nextButton);
        subscriptionPage.selectUserRegKeyIfRequired(userRegKeyLabel);
    }

    public boolean is LoginPageDisplayed() {
        try {
            waiters.waitForElementToBeDisplay(userNameSelector);
            return true;
        } catch (Exception e) {
            return false;
        }
    }

    public boolean verifySuccessfulSignOutMessage() {

waiters.waitForElementToBeDisplay(By.xpath(String.format(SIGN_OUT_LOCATOR,
"You have")));
        if
(waiters.isElementDisplayed(By.xpath(String.format(SIGN_OUT_LOCATOR,
                "You have successfully signed off your single sign-on
session."))) ||

waiters.isElementDisplayed(By.xpath(String.format(SIGN_OUT_LOCATOR,
                        "You have been successfully signed out."))))
        {
            LOG.info("Logout message is expected. Url: " +
Driver.getDriver().getCurrentUrl());
            return true;
        }
        else {
           return false;
        }
    }

    public String getLogoutUrl() {
        return Driver.getDriver().getCurrentUrl();
    }

    public void clickSignBackInLink() {
        clickOnElement(signBackInLink);
    }
```

Потім ми можемо використовувати ці сторінки на рівнях проксі-класах визначає кроки, які потім пов’язуються зі сценаріями користувача мовою Gherkin.

Приклад такого класу:

```
      public class  UsersOperations_sd {

    static final Logger LOG = LoggerFactory.getLogger(
UsersOperations_sd.class);

    private  RestClient  RestClient = new  RestClient();

    private AdminClientPage adminClientPage =
PageFactory.initElements(Driver.getDriver(), AdminClientPage.class);

    private AdminPage adminPage =
PageFactory.initElements(Driver.getDriver(), AdminPage.class);

    private AdminTRPage adminTRPage =
PageFactory.initElements(Driver.getDriver(), AdminTRPage.class);

    private AdminUserPage adminUserPage =
PageFactory.initElements(Driver.getDriver(), AdminUserPage.class);

    private Login  login Page = PageFactory.initElements(Driver.getDriver(),
Login .class);

    private BasePage basePage = new BasePage();

    Scenario scenario;

    public final static String  _CLIENT_DETAILS = " _CLIENT_DETAILS";

    private final static String  _USER_ID = " _USER_ID";

    private String getShortRandomValue() {
        return UUID.randomUUID().toString().replaceAll("-", "").substring(0,
8);
```

```
    }

    @Before
    public void beforeScenario(Scenario scenario) {
        this.scenario = scenario;
    }

    @When("I open {string} client")
    public void iOpenClientByName(String clientName) {
        ClientDto clientDto = getClientDto(clientName);
        if (clientDto == null) {
            throw new RuntimeException(String.format("Unable to open Client
'%s'", clientName));
        }
        adminTRPage.openClientViaDirectUrl(clientDto.getClientName());

adminPage.getAdminSideBarItemAfterWaitForIsDisplayed(clientDto.getClientName(
));
        scenario.write("Client Name:" + clientDto.getClientName());
    }

    @Given("I logged in to WCO as static   user")
    public void iEnsureIHaveLoggedInAs StaticUser() {
        login
Page.loginSpecifyingRegKeyLabel(TestContext.getInstance().getUser(),

TestContext.getInstance().getPassword(),

TestContext.getInstance().getUserRegKeyLabel());
        login Page.switchToFrame();
    }

    @Given("I ensure that client \"([^\"]*)\" is generated and stored")
    public void iEnsureClientIsGeneratedAndStored(String clientName,
Map<String, String> specificDetails) {
        ClientDto clientDto = getClientDto(clientName);
        if (clientDto == null) {
            String trustId = specificDetails.get("Trust ID");
            adminClientPage.clickCreateClient();
```

```
iFillInClientSubscriptionInformationWithSpecificDetailsAndStoreIt(clientName,
trustId, specificDetails);
            adminClientPage.clickOnAdminSettingsButton("Next >>");
            adminClientPage.fillInObligotaryClientSubscriptionDetails();
            adminClientPage.clickOnAdminSettingsButton("Next >>");
            adminClientPage.fillInObligotaryResolutionSettings();
            adminClientPage.clickOnAdminSettingsButton("Next >>");
            adminClientPage.clickOnAdminSettingsButton("Create client >>");
            if (!adminClientPage.isStatusMessageContains("Client created
successfully")) {
                storeClientDto(clientName, null);
                throw new RuntimeException("Unable to create Client via UI");
            }

TestContext.getInstance().addClientToClientListForFutureDeletionInAfterHook(g
etClientDto(clientName));
            LOG.info("Client is created: " +
getClientDto(clientName).getClientName());
        }
        else {
            LOG.info("Client is already created: " +
getClientDto(clientName).getClientName());
        }
    }
```

```
    @Then("I fill in client Details information with Trust ID \"([^\"]*)\"
and specific details")
    public void
iFillInClientSubscriptionInformationWithSpecificDetails(String trustId,

Map<String, String> specificDetails) {
        iFillInClientSubscriptionInformationWithSpecificDetailsAndStoreIt(
_CLIENT_DETAILS, trustId, specificDetails);
    }

    private void
iFillInClientSubscriptionInformationWithSpecificDetailsAndStoreIt(String
clientName, String trustId,

Map<String, String> specificDetails) {
        String defaultValue = "999999999";
        String shortRandomValue = getShortRandomValue();
        String emailAddress = specificDetails
            .getOrDefault("Email", String.format("bddtestuser_%s@tr.com",
shortRandomValue))
            .replace("replaceRandomNumber", shortRandomValue)
            .replace("mockUserEmail",
Optional.ofNullable(getStoredEmail("mockUserEmail")).orElse(""));

        ClientDto clientDto =
            ClientDto.NewClientBuilder()
                    .trustId(trustId)
                    .clientName(specificDetails.getOrDefault("Client name",

"zFrontEndClient-replaceRandomNumber")

.replace("replaceRandomNumber", shortRandomValue))
                    .salesforceId(specificDetails.getOrDefault("Salesforce
ID", defaultValue))
                    .numberOfUsers(specificDetails.getOrDefault("Number of
users", defaultValue))
                    .numberOfSearches(
                        specificDetails.getOrDefault("Number of searches",
defaultValue))
                    .numberOfWcOgsSearches(specificDetails.getOrDefault(
                        "Number of World Check and Watchlist OGS Searches",
defaultValue))
                    .numberOfMcOgsSearches(specificDetails
                                                .getOrDefault("Number of
Media Check OGS searches",
```

```
                                                                defaultValue))
                    .numberOfTotalCases(specificDetails.getOrDefault("Number
of Total Cases", defaultValue))
                    .complianceLeaderName(
                        specificDetails.getOrDefault("Compliance leader
name", "Nucky"))
                    .complianceLeaderEmail(
                        specificDetails.getOrDefault("Compliance leader
email", emailAddress))
                    .clientAdminFirstName(
                        specificDetails.getOrDefault("Client admin first
name", defaultValue))
                    .clientAdminLastName(
                        specificDetails.getOrDefault("Client admin last
```

```
name", defaultValue))
                    .clientAdminEmail(
                        specificDetails.getOrDefault("Client admin email",
emailAddress))
                    .clientAdminUserName(
                        specificDetails.getOrDefault("Client admin user
name", emailAddress))
                    .build();
        adminClientPage.fillInClientDetailsViaDto(clientDto);
        storeClientDto(clientName, clientDto);
    }

    @When("I validate that User Status for \"([^\"]*)\" email is
\"([^\"]*)\"")
    public void iValidateUserStatus(String storedEmailKey, String userStatus)
{
        String email = getStoredEmail(storedEmailKey);
        String  RequestId = AccelusDatabaseHelper.getInstance().get
RequestId(email);
        assertThat(" _REQUEST_ID should be not null.",  RequestId,
notNullValue());
        Awaitility.await().with()
                .pollInterval(Duration.ONE_SECOND)
                .atMost(new Duration(

Integer.parseInt(TestContext.getInstance().getProperty("PENDING_USER_TIMEOUT"
)),
                        TimeUnit.MINUTES)
                )
                .conditionEvaluationListener(condition ->
                        LOG.info("Wait User Status in DB. Expected: " +
userStatus))
                .until(() ->
AccelusDatabaseHelper.getInstance().getUserStatus( RequestId),
equalTo(userStatus));
    }

    @When("I clear User Data for \"([^\"]*)\" email")
    public void iClearUserData(String storedEmailKey) {
        String email = getStoredEmail(storedEmailKey);
        AccelusDatabaseHelper.getInstance().clearUserData(email);
    }

    @Then("I verify that User Name field is \"([^\"]*)\"$")
    public void iVerifyUserNameField(String userName) {
        assertThat("UserName is wrong.", adminUserPage.getUsername(),
```

```
is(userName));
    }

    @Then("I validate that   user role is \"([^\"]*)\" as \"([^\"]*)\"$")
    public void iValidateThat UserRoleIs(String expected UserRole, String
additionalExplanation) {
        String  UserId = get UserId();
         RestClient.validateUserRole( UserId, expected UserRole,
additionalExplanation);
    }

    @And("I generate test \"([^\"]*)\" email as \"([^\"]*)\"$")
    public void iGenerateTestEmail(String storedEmailKey, String
emailTemplate) {
        String emailAddress = emailTemplate.replace("replaceRandomNumber",
getShortRandomValue());
        storeEmail(storedEmailKey, emailAddress);
    }
```

```
    @And("I successfully created a user with firstname \"([^\"]*)\" lastname
\"([^\"]*)\" and \"([^\"]*)\" email and roles$")
    public void iCreatedAUserWithFirstnameLastnameAndEmailAndRoles(String
firstName, String lastName,
                                                                   String
emailKey, List<String> roles) {
        String userName = getStoredEmail(emailKey);
        stabilizedTrAdminOperationsPage
            .createUser(firstName, lastName, getStoredEmail(emailKey), roles,
true);
        //store reg key
        storeEmail(firstName + " " + lastName,
FspDB.getRegKeyForEmailAndLastName(userName, lastName));
    }

    @And("I successfully created a user with firstname \"([^\"]*)\" lastname
\"([^\"]*)\" and \"([^\"]*)\" not stored email and roles$")
    public void
iCreatedAUserWithFirstnameLastnameAndNotStoredEmailAndRoles(String firstName,
String lastName,

String email, List<String> roles) {
        stabilizedTrAdminOperationsPage
                .createUser(firstName, lastName, email, roles, true);
    }

    @And("I successfully created a user with firstname \"([^\"]*)\" lastname
\"([^\"]*)\" and \"([^\"]*)\" email and without roles$")
    public void iCreatedAUserWithFirstnameLastnameAndEmailWithoutRoles(String
firstName, String lastName,
                                                                   String
emailKey) {
        String userName = getStoredEmail(emailKey);
        stabilizedTrAdminOperationsPage
            .createUser(firstName, lastName, userName,
Collections.emptyList(), true);
        //store reg key
        storeEmail(firstName + " " + lastName,
FspDB.getRegKeyForEmailAndLastName(userName, lastName));
        scenario.write("User name:" + userName);
    }

    @And("I create a user with firstname \"([^\"]*)\" lastname \"([^\"]*)\"
and static user email and without roles$")
    public void iCreateUserWithFirstnameLastnameAndEmailWithoutRoles(String
```

```
firstName, String lastName) {
        String emailKey =
TestContext.getInstance().getProperty("STATIC_USERNAME");
        stabilizedTrAdminOperationsPage
                .fillNewUserFields(firstName, lastName, emailKey,
Collections.emptyList(), true);
    }

    @When("I create a user with firstname \"([^\"]*)\" lastname \"([^\"]*)\"
email \"([^\"]*)\" and without roles")
    public void
i_create_a_user_with_firstname_lastname_email_and_without_roles(String
firstName, String lastName, String email) {
        stabilizedTrAdminOperationsPage
                .fillNewUserFields(firstName, lastName, email,
Collections.emptyList(), true);
    }
```

```
    @When("I create a user with firstname \"([^\"]*)\" lastname \"([^\"]*)\"
email \"([^\"]*)\" and roles")
    public void
i_create_a_user_with_firstname_lastname_email_and_roles(String firstName,
String lastName, String email, List<String> roles) {
        stabilizedTrAdminOperationsPage
                .fillNewUserFields(firstName, lastName, email, roles, true);
    }

    @Then("I validate that a user is not created with firstname \"([^\"]*)\"
lastname \"([^\"]*)\" and \"([^\"]*)\" email and roles$")
    public void
iDidntCreatedAUserWithFirstnameLastnameAndEmailAndRoles(String firstName,
String lastName,

String emailKey, List<String> roles) {
        stabilizedTrAdminOperationsPage
                .createUserFailed(firstName, lastName,
getStoredEmail(emailKey), roles, true);
    }

    @And("I successfully added roles for user \"([^\"]*)\"$")
    public void iAddedRolesForUser(String userName, List<String> roles) {

stabilizedTrAdminOperationsPage.addRolesForUserAndBackToUsers(userName,
roles);
    }

    @And("I successfully added role for user \"([^\"]*)\"$")
    public void iAddedRoleForUser(String userName, List<String> roles) {
        stabilizedTrAdminOperationsPage.addRolesForUser(userName, roles);
    }

    @And("I validate that roles are not added for user \"([^\"]*)\"$")
    public void rolesForUserAreNotAdded(String userName, List<String> roles)
{
        stabilizedTrAdminOperationsPage.addRolesForUserFailed(userName,
roles);
    }

    @And("I successfully unassigned roles for user \"([^\"]*)\"$")
    public void iRemovedRolesForUser(String userName, List<String> roles) {

stabilizedTrAdminOperationsPage.removeRolesForUserAndBackToUsers(userName,
roles);
    }
```

```
    }

    @And("I successfully unassigned role for user \"([^\"]*)\"$")
    public void iRemovedRoleFromUser(String userName, List<String> roles) {
        stabilizedTrAdminOperationsPage.removeRolesForUser(userName, roles);
    }

    public ClientDto getClientDto(String clientName) {
        return basePage.getObjectFromSession(clientName, ClientDto.class);
    }

    public void storeClientDto(String clientName, ClientDto clientDto) {
        basePage.storeObjectToSession(clientName, clientDto);
    }

    public String getStoredEmail(String storedEmailKey) {
        return basePage.getObjectFromSession(storedEmailKey, String.class);
    }
```

```
    public void storeEmail(String emailKey, String emailAddress) {
        basePage.storeObjectToSession(emailKey, emailAddress);
    }

    @Then("^I successfully delete a user \"([^\"]*)\"$")
    public void iSuccessfullyDeleteAUser(String userName) {
        stabilizedTrAdminOperationsPage.deleteUser(userName);
    }
```

Методи цього класу, які прив’язані до певних певних кодових фраз(степів або кроків) ми можемо використовувати у юзер сценаріях.

Приклад такого сценарію:

```
      @logout_url_change
Feature: Logout URL change [WC1-28057]
    As a WC1 User
    I want to be able to log out from WC1
    So that I do not stay permanently logged in

    @ _extra_tracked_by_petros @logout_url_change1
    Scenario: logout as   TR Admin
        Given I logout from app
        And I logged in to WCO as static user
        When I refresh the page
        When I am logging out
        Then I see success sign out message
        And I verify logout page URL
        #When I click on Sign back in link
        #Then I see login page

    @ _extra_tracked_by_petros @logout_url_change2
    Scenario: logout as OnePass user
        Given I have logged in to WCO as "ALL_ROLE_USER"
        When I am logging out
        Then I see success sign out message
        And I verify logout page URL
        #When I click on Sign back in link
        #Then I see login page
```

Тепер, коли зовнішня частина нашого фреймворку реалізована, ми можемо перейти до створення розділу, який відповідатиме за тестовий API.

3.3 API частина

По-перше, нам потрібно зібрати всі наші кінцеві точки в одному місці для легкого редагування та додавання нових.

Тепер давайте перейдемо до класу, де ми будемо викликати ці сценарії і потім перейдемо, де буде проходити тестування API.

```
        public class RestClient {

    private static final Header AUTORIZATION_HEADER_SYSTEM = new
Header("Authorization", "SYSTEM");

    private  Header userNameHeader;
    private  Header authorizationHeader;

    public AccelusRestClient(String userRegKey) {
        this.userNameHeader = new Header(USER_NAME_HEADER_KEY, userRegKey);
        this.authorizationHeader = new Header("Authorization", userRegKey);
        RestAssured.baseURI =
TEST_CONTEXT.getProperty("ACCELUS_SECURITY_REST_BASE_URI");
    }

    public void setHeaders(String headersRegKey){
        this.userNameHeader = new Header(USER_NAME_HEADER_KEY,
headersRegKey);
        this.authorizationHeader = new Header("Authorization",
headersRegKey);
    }

    public Response getUserDetails(String regKey) {
        RestAssured.basePath = V4.getEndpoint();
        String url = String.format(GET_USER.getEndpoint(), regKey);
        Headers headers = new Headers(AUTORIZATION_HEADER_SYSTEM,
JSON_HEADER);

        return doGetRestCall(url, headers);
    }

    public Response getAccelusUserDetails(String regKey) {
        RestAssured.basePath = V2.getEndpoint();
        String url = String.format(GET_ACCELUS_USER_DETAILS.getEndpoint(),
regKey);
        Headers headers = new Headers(AUTORIZATION_HEADER_SYSTEM,
JSON_HEADER);

        return doGetRestCall(url, headers);
    }

    public Response getAllUsersWithPagination(Long page, Long pageSize) {
        String accelusSecurityRestHost =
TEST_CONTEXT.getProperty("ACCELUS_SECURITY_REST_BASE_URI");
        String template = "%s/v2/user?page=%d&size=%d";
        String url = String.format(template, accelusSecurityRestHost, page,
```

```
public class LocationSearchAT extends BaseAT {

    HttpStatus successStatusCode = OK;
    private String rdpUsername = TestContext.getInstance().getUser();
    private String rdpPassword = TestContext.getInstance().getPassword();
    private String rdpRegKey = TEST_CONTEXT.getProperty("RDP_REGKEY");
    private String searchBy;
    private String filter;
    private List<String> relatedLocations;

    @After
    public void tearDown() {
        accelusRestClient.setHeaders(getTRAdminUser());
    }

    @Before
    public void prepare() throws EncoderException {
        LOG.info("############### Login to wco.");
        AccelusUtils.accelusPingLogin(rdpUsername, rdpPassword, rdpRegKey);
        accelusRestClient.setHeaders(rdpRegKey);
    }


    @Test
    public void getLocationBySearchTerm() {
        searchBy = "location_id";
        filter = "A-88888888";
        relatedLocations = Collections.emptyList();
        String searchName = "Internal TR Fixed-Interest Trading Application";

        GetLocationsByTerm getLocationBody = new GetLocationsByTerm(searchBy,
filter, relatedLocations);
        searchBy = getLocationBody.getSearchBy();
        LOG.info("############### Search location by  " + searchBy);
        Response locationsByLocationId =
accelusRestClient.getLocationsByTerm(getLocationBody);
        accelusRestClient.validateResponseLocations(locationsByLocationId,
filter, searchBy, successStatusCode.value());

        getLocationBody.setSearchBy("location_name");
        getLocationBody.setFilter(searchName);

        searchBy = getLocationBody.getSearchBy();
        filter = getLocationBody.getFilter();
        LOG.info("############### Search location by " + searchBy);
        Response locationsByName =
accelusRestClient.getLocationsByTerm(getLocationBody);
        accelusRestClient.validateResponseLocations(locationsByName, filter,
searchBy, successStatusCode.value());

        getLocationBody.setSearchBy("all_fields");
        searchBy = getLocationBody.getSearchBy();
        LOG.info("############### Search location by " +
getLocationBody.getSearchBy());
        Response locationsByAllFields =
accelusRestClient.getLocationsByTerm(getLocationBody);
```

```
        assertThat(locationsByAllFields.getStatusCode(),
is(INTERNAL_SERVER_ERROR.value()));
    }

    @Test
    public void getLocationBySearchTermWithRelatedLocation() {
        searchBy = "location_id";
        filter = "A-00166143";
        String searchName = "Reliance Industries";
        relatedLocations = Arrays.asList("A-00105574", "A-00571044");

        GetLocationsByTerm getLocationBody = new GetLocationsByTerm(searchBy,
filter, relatedLocations);
        LOG.info("############### Search location by  " + searchBy);
        Response locationsByLocationId =
accelusRestClient.getLocationsByTerm(getLocationBody);

accelusRestClient.validateResponseLocationWithRelatedLocation(locationsByLoca
tionId, filter, searchBy,
                relatedLocations, successStatusCode.value());

        getLocationBody.setSearchBy("location name");
        getLocationBody.setFilter(searchName);

        searchBy = getLocationBody.getSearchBy();
        filter = getLocationBody.getFilter();
        LOG.info("############### Search location by " +
getLocationBody.getSearchBy());
        Response locationsByName =
accelusRestClient.getLocationsByTerm(getLocationBody);
```

```
accelusRestClient.validateResponseLocationWithRelatedLocation(locationsByName
, filter, searchBy,
                relatedLocations, successStatusCode.value());

        getLocationBody.setSearchBy("all fields");
        searchBy = getLocationBody.getSearchBy();
        LOG.info("############### Search location by " +
getLocationBody.getSearchBy());
        Response locationsByAllFields =
accelusRestClient.getLocationsByTerm(getLocationBody);

accelusRestClient.validateResponseLocationWithRelatedLocation(locationsByAllF
ields, filter, searchBy,
                relatedLocations, successStatusCode.value());
    }
}
```

Пам’ятайте, що у нас достатньо вмісту запитів і відповідей зокрема, і для кращого вирішення цих проблем варто створити класи відповідні цим установам. В даний час структура має два типи реалізації даних власність Через анотації Loombok.

```
@Getter
@AllArgsConstructor
@NoArgsConstructor
@Setter
@Builder
@ToString
@EqualsAndHashCode

public class RdpLicense {
    private String productId;
    private String productName;
}
```

І стандартний метод реалізації класу JAVA

```
)
public class UserRegistrationResponse {
    private final Status status;
    private final FirstTimeLoginToken firstTimeLoginToken;
    private final String registrationKey;
    private final boolean success;
    private final String failureReason;
    private final String failureData;

    private UserRegistrationResponse() {
        this.status = null;
        this.firstTimeLoginToken = null;
        this.registrationKey = null;
        this.success = false;
        this.failureReason = null;
        this.failureData = null;
    }

    public UserRegistrationResponse(Status status, String registrationKey,
FirstTimeLoginToken firstTimeLoginToken, boolean success, String
failureReason, String failureData) {
        this.status = status;
        this.firstTimeLoginToken = firstTimeLoginToken;
        this.registrationKey = registrationKey;
        this.success = success;
        this.failureReason = failureReason;
        this.failureData = failureData;
    }

    public UserRegistrationResponse(UserRegistrationResponse.Builder builder)
{
        this.status = builder.getStatus();
        this.firstTimeLoginToken = builder.getFirstTimeLoginToken();
        this.registrationKey = builder.getRegistrationKey();
        this.success = builder.isSuccess();
        this.failureReason = builder.getFailureReason();
        this.failureData = builder.getFailureData();
    }

    public String getRegistrationKey() {
        return this.registrationKey;
    }

    public Status getStatus() {
```

```
        return this.status;
    }

    public FirstTimeLoginToken getFirstTimeLoginToken() {
        return this.firstTimeLoginToken;
    }

    public boolean isSuccess() {
        return this.success;
    }

    public String getFailureReason() {
        return this.failureReason;
    }

    public String getFailureData() {
        return this.failureData;
    }

    public boolean equals(Object o) {
        if (this == o) {
            return true;
        } else if (o != null && this.getClass() == o.getClass()) {
            UserRegistrationResponse response = (UserRegistrationResponse)o;
            return this.success == response.isSuccess() && this.status ==
response.getStatus() && Objects.equals(this.firstTimeLoginToken,
response.getFirstTimeLoginToken()) && Objects.equals(this.registrationKey,
response.getRegistrationKey()) && Objects.equals(this.failureData,
response.getFailureData()) && Objects.equals(this.failureReason,
response.getFailureReason());
        } else {
            return false;
        }
    }

    public int hashCode() {
        return Objects.hash(new Object[]{this.status,
this.firstTimeLoginToken, this.registrationKey, this.success,
this.failureReason, this.failureData});
    }

    public String toString() {
        return ToStringBuilder.reflectionToString(this,
ToStringStyle.SHORT_PREFIX_STYLE);
    }
```

```
    public static final class Builder {
        private Status status;
        private FirstTimeLoginToken firstTimeLoginToken;
        private String registrationKey;
        private boolean success;
        private String failureReason;
        private String failureData;

        public Builder() {
        }

        public Builder(UserRegistrationResponse userRegistrationResponse) {
            this.status = userRegistrationResponse.getStatus();
            this.firstTimeLoginToken =
userRegistrationResponse.getFirstTimeLoginToken();
            this.registrationKey =
userRegistrationResponse.getRegistrationKey();
            this.success = userRegistrationResponse.isSuccess();
            this.failureReason = userRegistrationResponse.getFailureReason();
            this.failureData = userRegistrationResponse.getFailureData();
        }

        public UserRegistrationResponse build() {
            return new UserRegistrationResponse(this);
        }

        public String getRegistrationKey() {
            return this.registrationKey;
        }

        public Status getStatus() {
            return this.status;
        }

        public FirstTimeLoginToken getFirstTimeLoginToken() {
            return this.firstTimeLoginToken;
        }

        public boolean isSuccess() {
            return this.success;
        }

        public String getFailureReason() {
            return this.failureReason;
        }

        public String getFailureData() {
            return this.failureData;
        }

        public UserRegistrationResponse.Builder withStatus(Status status) {
            this.status = status;
            return this;
        }

        public UserRegistrationResponse.Builder
withFirstTimeLoginToken(FirstTimeLoginToken firstTimeLoginToken) {
            this.firstTimeLoginToken = firstTimeLoginToken;
            return this;
        }
```

```
        public UserRegistrationResponse.Builder
withFirstTimeLoginToken(FirstTimeLoginToken firstTimeLoginToken) {
            this.firstTimeLoginToken = firstTimeLoginToken;
            return this;
        }

        public UserRegistrationResponse.Builder withRegistrationKey(String
registrationKey) {
            this.registrationKey = registrationKey;
            return this;
        }

        public UserRegistrationResponse.Builder withSuccess(boolean success)
{
            this.success = success;
            return this;
        }

        public UserRegistrationResponse.Builder withFailureReason(String
failureReason) {
            this.failureReason = failureReason;
            return this;
        }

        public UserRegistrationResponse.Builder withFailureData(String
failureData) {
            this.failureData = failureData;
```

Отже, наші тести пройшли успішно, і ми можемо спробувати запустити його локально, щоб перевірити їхню функціональність.

Так виглядає коли наші тести працюють успішно і без помилок.

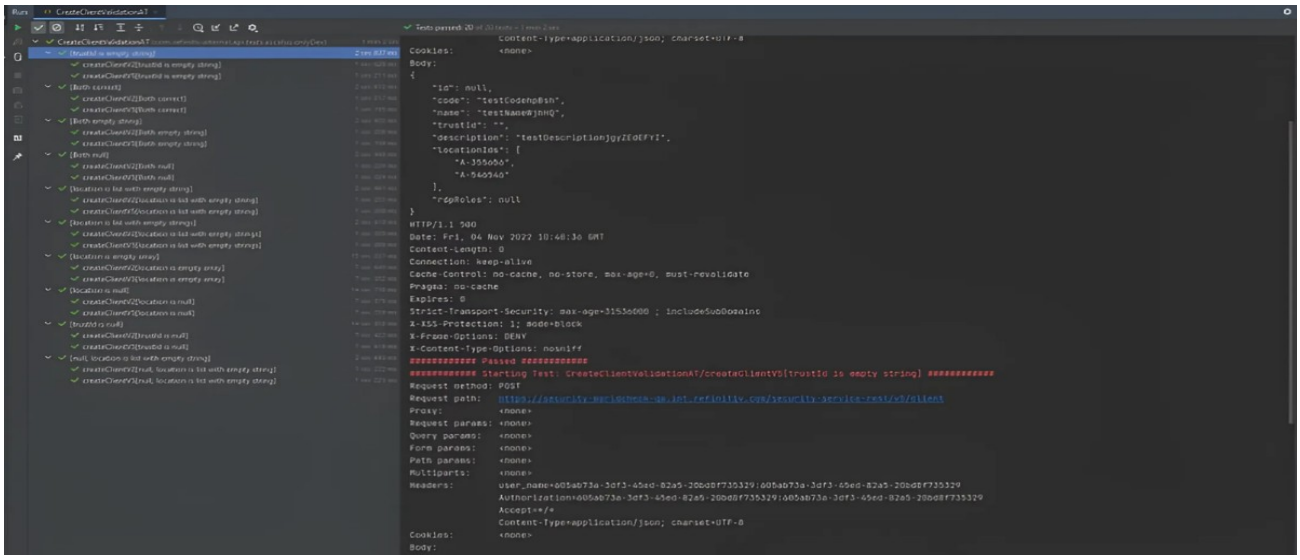


Рисунок 3.2 - Успішне виконання

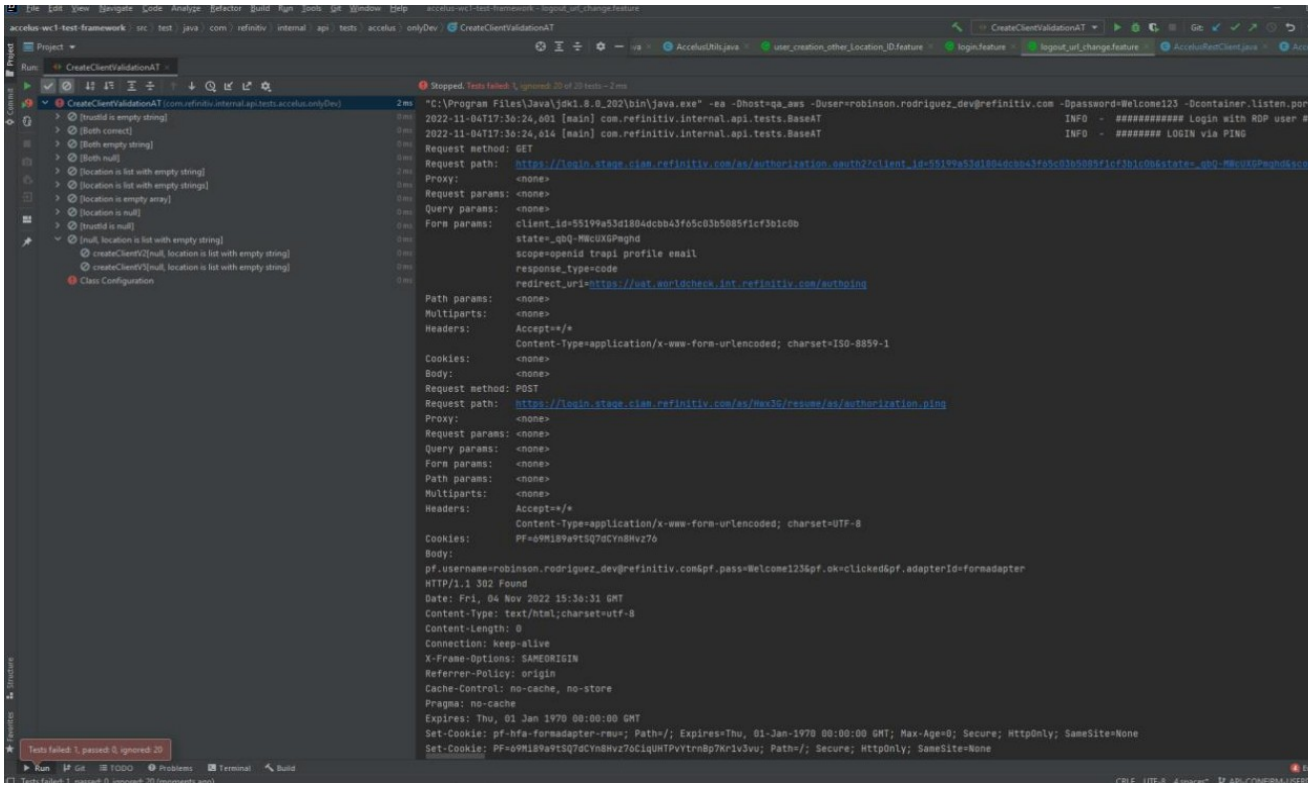


Рисунок 3.3 – Невдале тестування

Платформа автоматизованого тестування, яку ми створили, призначена для виконання таких завдань:

● Додавання нових класів функцій і кінцевих точок, щоб розширити функції продукту;

● Автоматизувати виконання тестів, формування розуміння для користувача;

● Повідомлення про результати тестування;

● Збереження історій тестів, їхні успіхи та місцеположення конкретних помилок, а також історія змін конфігурації автоматизації;

● Відстежувати продуктивність окремих модулів програми;

● Надає доступ до тестових сценаріїв. І дає змогу редагувати або оптимізувати для обмеженої групи людей;

Користувач проєкту, що використовуватиме цей продукт для підвищення кінцевої якості проєкту значно заощадить час та ресурси. Усі тестові випадки можна виконувати без підключення до місця розташування, або платформи, що значно спрощує життя тестувальника. Продукт містить тестові сценарії для всіх критичних функцій проєкту, що полегшує оцінку. Загальний рівень оцінки стану проєкту полягає в пошуку конкретного місця поломки в разі проблеми. Це практично, тому що користувачі можуть автоматично бачити всі передумови та виконані кроки перед error.loc.

ВИСНОВКИ

Сьогодні кожен продукт і кожен клієнт шукають шляхи зниження витрат до мінімального рівня і підняття рівня якості кінцевого продукту до максимального. Проте, не всі знають, як цього досягти шляхом запровадження відповідних підходів і методів тестування, а також у частковому випадку автоматизації. Тож, ідеальним варіантом у цьому випадку створити правильну стратегію тестування та використати правильні застосунки та спеціалістів.

У кваліфікаційній роботі проаналізовано найважливіші питання про те, яким чином підійти до побудови тесту, структури та тестування всього проєкту: перегляд вимог до продукції та розробка загальної стратегії тестування на ранніх стадіях SDLC; співпраця всієї команди розробників, тестувальників із замовниками; план, як оптимізувати та покращити створені тест плани, стратегії та сценарії.

Для розгортання тестової системи створено базову модель автоматизованої системи тестування з відкритим кодом, яку реалізовано використовуючи наступні правила: підвищення кінцевої якості продукту; універсальність та легкість модернізації; автоматизовані й ручні тести.

Створена система автоматизованого тестування призначена для виконання наступних завдань:

- додавання нових класів функцій і кінцевих точок розширення функції продукту;
- автоматизування виконання тестів;
- повідомлення про результати тестування;
- збереження історії тестів, їхні успіхи та місцеположення конкретної помилки, а також історії змін конфігурації автоматизації;
- відстеження продуктивності окремих модулів програми;
- доступ до тестових сценаріїв для редагування або оптимізації обмеженої кількості людей.

Користувачі проєкту використовуватимуть цей продукт для підвищення кінцевої якості проєкту, що значно заощадить час і гроші. Усі тестові випадки

можна виконувати без прив’язки до місця розташування, що значно спрощує життя тестувальника. Продукт містить тестові сценарії для всіх критичних функцій проєкту, що полегшує оцінку несправностей. Загальний рівень оцінки стану проєкту полягає в пошуку конкретного місця поломки в разі проблеми, тому що користувачі можуть автоматично бачити всі передумови та виконані кроки перед помилкою.

Дане дослідження допоможе підвищити кінцеву якість продукту на ранніх стадіях розробки, а також зменшити витрати на впровадження правильних шляхів для подальших удосконалень.

Перелік посилань

1. НАЙКРАЩИЙ інструмент тестування вебсайтів (тестування вебдодатків) 2024 (guru99.com) – https://www.guru99.com/uk/top-20-web-testing-tools.html.

2. Тестування вебпроєктів: основні етапи та поради - QALight – https://qalight.ua/baza-znaniy/testuvannya-veb-proektiv-osnovni-etapi-ta-poradi/.

3. Як тестувати вебсайт: основні етапи і поради - https://brainlab.com.ua/uk/blog-uk/yak-testuvati-veb-sayt-osnovn-etapi-poradi.

4. Тестування вебдодатків: як перевірити вебсайт? (guru99.com) - https://www.guru99.com/uk/web-application-testin.html

5. REDSTONE :: 6 видів тестування web-сайтів: особливості та з чого почати - https://redstone.media/vydy-testuvannya-websaytiv

6. Тестування вебсайтів: види, методи і цілі - Блог VOLL - https://voll.com.ua/uk/blog/kak-testirovat-veb-sajt-osnovnye-vidy-i-celi-testirovaniya

7. Підручники для ознайомлення з тестуванням для вузів онлайн - https://pidru4niki.com/1594102439093/psihologiya/metod_testuvannya

8. Статичне та динамічне тестування: основні відмінності та практичні приклади - Блог Mate academy - https://mate.academy/blog/qa/static-dynamic-testing/

9. Чек лист для тестування сайту та мобільного додатка - https://avada-media.ua/ua/services/chek-list-dlya-testirovaniya-sayta-i-mobilnogo-prilozheniya/

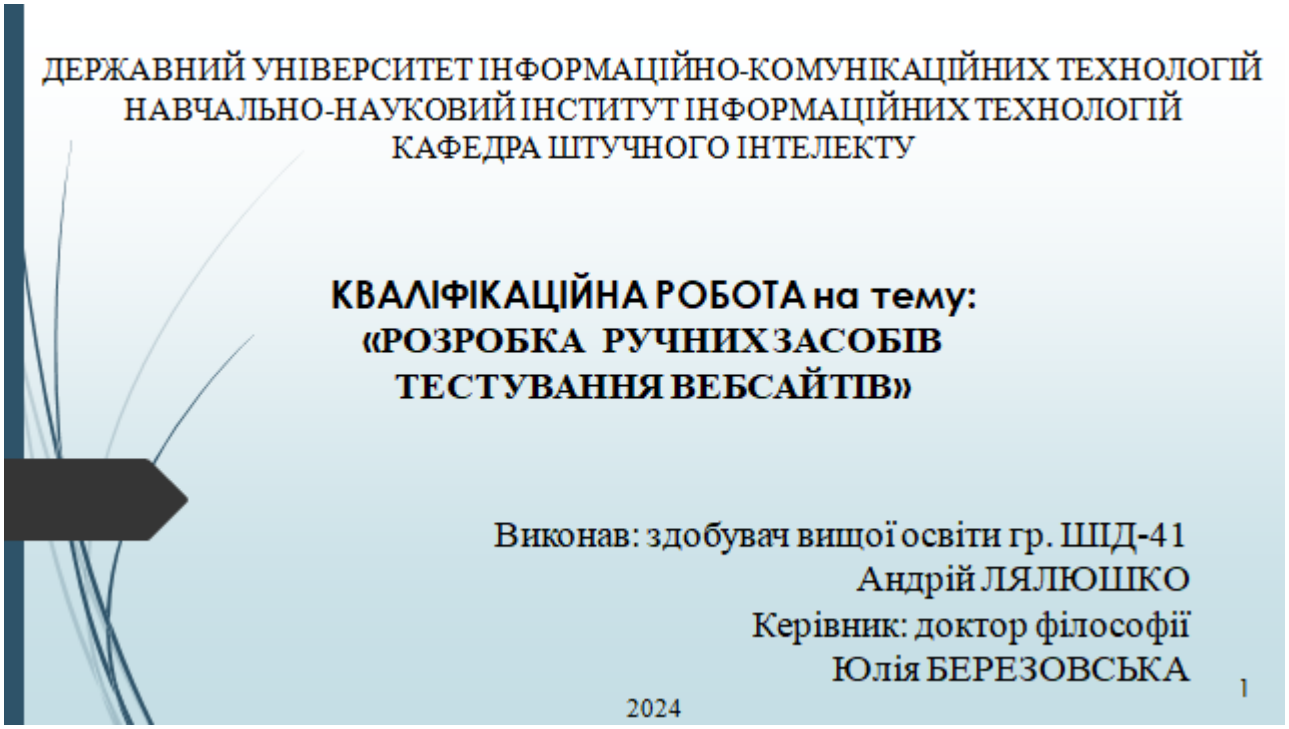
10. What is Selenium? Introduction to Selenium Automation Testing - https://www.guru99.com/introduction-to-selenium.html

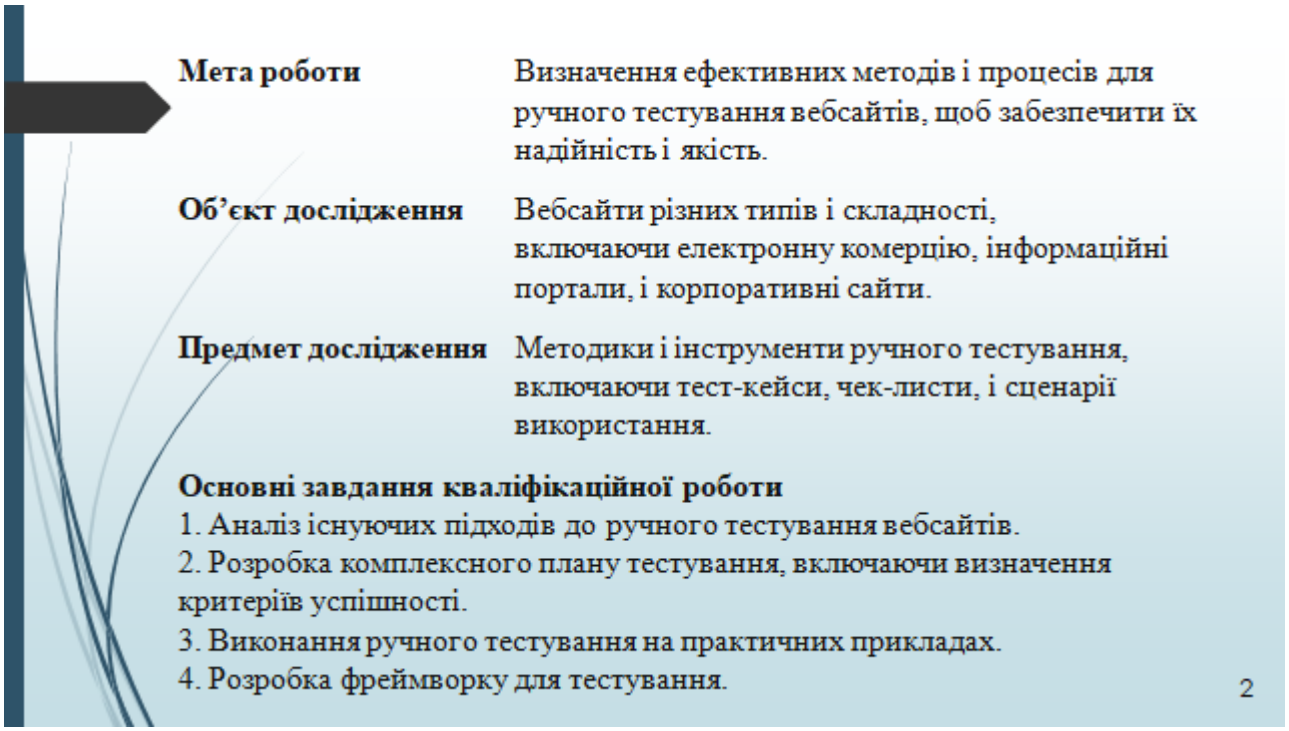
11. Understanding Cloud Services: What is cloud computing? - https://www.businesstechweekly.com/operational-efficiency/cloud-computing/what-is-cloud-computing-what-is-the-cloud/

12. Top 4 Embedded Operating Systems with Examples - https://blog.felgo.com/embedded/embedded-operating-systems

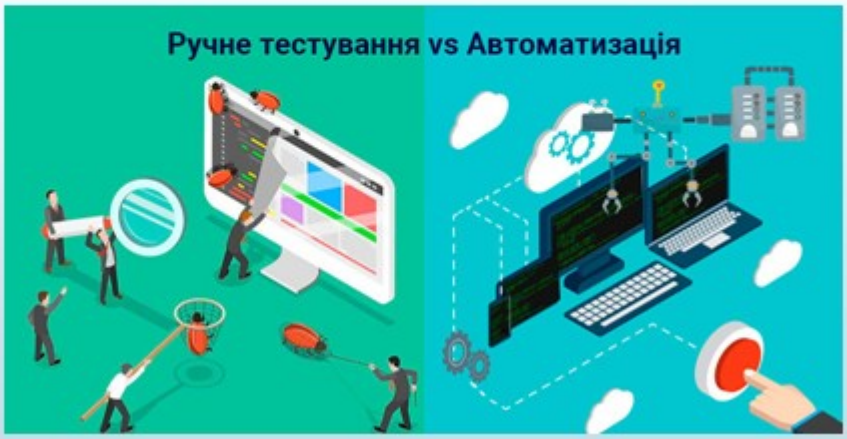
13. Understanding Cloud Services: What is cloud computing? - https://www.businesstechweekly.com/operational-efficiency/cloud-computing/what-is-cloud-computing-what-is-the-cloud/

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ





Відмінності ручного та автоматизованого тестувань



3

Ручне тестування

1. **Визначення**:
   1. Виконується вручну аналітиком із забезпечення якості (людиною).
   2. Тестувальник ретельно перевіряє функціональність, продуктивність, надійність та взаємодію програмного забезпечення з користувачами.

2. **Принципи**:
   1. Глибокий аналіз функціонала.
   2. Реалістичні сценарії використання.
   3. Чітка та зрозуміла документація.
   4. Творчий підхід.

3. **Переваги**:
   1. Гнучкість.
   2. Інтуїтивність.
   3. Ефективність при швидкому розвитку проєкту.

4

Автоматизоване тестування

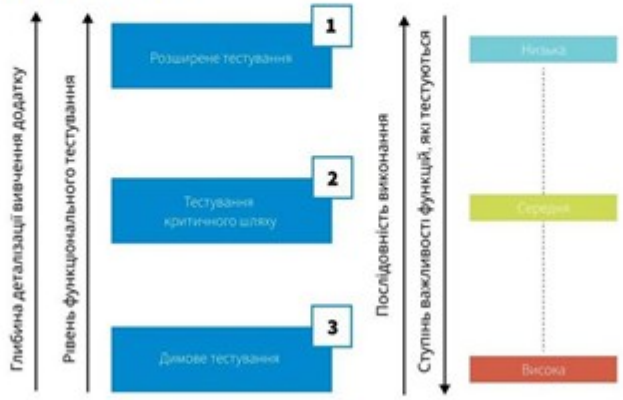
1. **Визначення**:
   1. Виконується за допомогою сценарію, коду та засобів автоматизації (комп'ютера).
   2. Тестувальник створює автоматизовані тест-кейси, які виконуються автоматично.

2. **Принципи**:
   1. Створення автоматизованих сценаріїв.
   2. Використання автоматизованих інструментів.
   3. Повторюваність та швидкість виконання.

3. **Переваги**:
   1. Надійність.
   2. Швидкість.
   3. Автоматизація рутинних завдань.

5

Класифікація за ступенем важливості функцій, що тестуються:



6

MANUAL TESTING

Enter your sub headline here

7

Загальні проблеми при ручному тестуванні

1. Невисока надійність.

2. Ризик помилок.

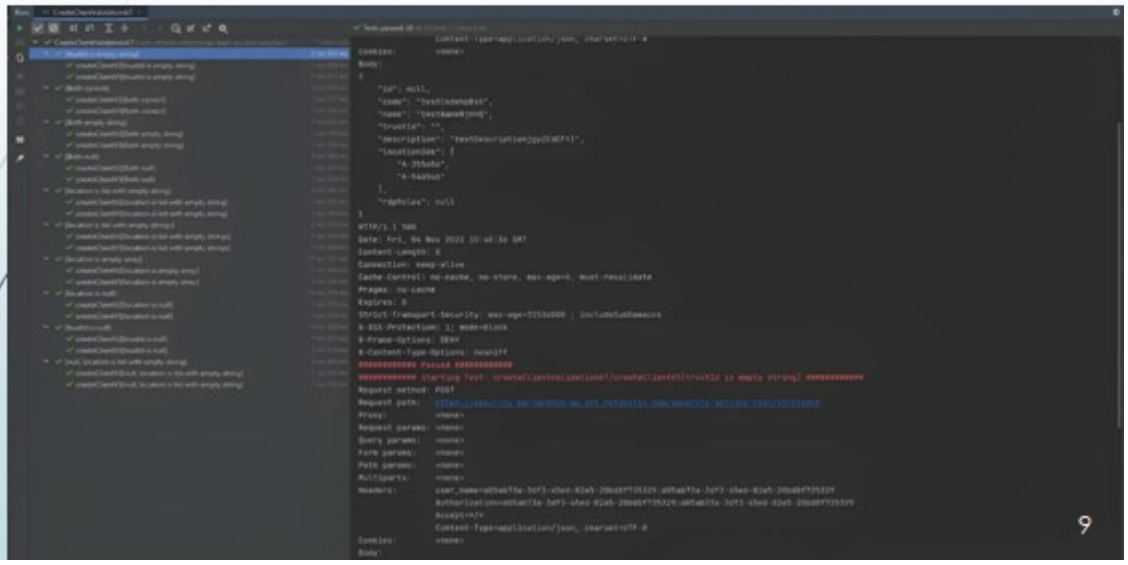
3. Часозатратність.

Можливі шляхи вирішення:

1. Створення тестової стратегії.
2. Написання тестового плану.
3. Обмеження кожного тестового кроку однією дією та перевіркою.

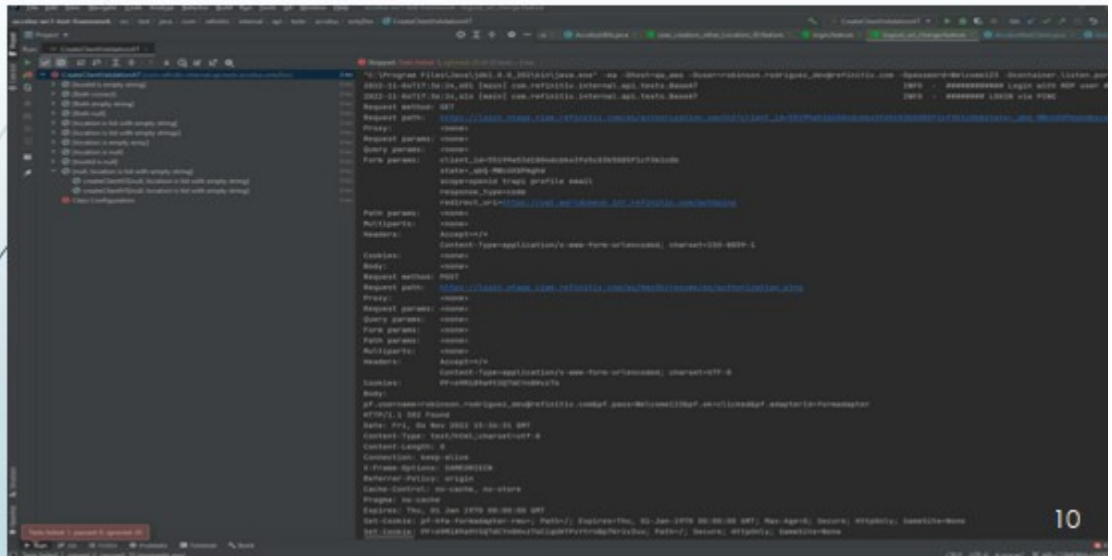
8

Вигляд автоматизованої системи, що не знайшла помилок на вебсайті



9

Вигляд системи, що виявила проблеми на вебсайті



10

Висновки

- Проаналізовано найважливіші питання, яким чином підійти до побудови тесту, структури та тестування всього проєкту: перегляд вимог до продукції та розробка загальної стратегії тестування на ранніх стадіях SDLC; співпраця всієї команди розробників, тестувальників із замовниками; план, як оптимізувати та покращити створені тест плани, стратегії та сценарії.
- Для розгортання тестової системи створено базову модель автоматизованої системи тестування з відкритим кодом, яку реалізовано використовуючи наступні правила: підвищення кінцевої якості продукту; універсальність та легкість модернізації; автоматизовані й ручні тести.
- Створено фреймворк для тестування за допомогою якого можна:
  - додавати нові класи функцій і кінцевих точок розширення функції продукту;
  - автоматизувати виконання певних тестів;
  - отримувати звітне повідомлення про результати тестування;
  - зберігати історії тестів, їхні успіхи та місцеположення конкретної помилки, а також історії змін конфігурації автоматизації;
  - відстежувати продуктивності окремих модулів програми;
  - отримати доступ до тестових сценаріїв.

11

Дякую за увагу

12