Міністерство освіти та науки України
Харківський національний університет радіоелектроніки

Факультет *Комп'ютерних наук*
(повна назва)
Кафедра *Системотехніки*
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

Рівень вищої освіти другий (магістерський)
(рівень вищої освіти)

Дослідження процесів контролю якості взаємодії користувачів з веб-додатком інтернет-магазину
(тема)

Виконав:
студент *2* курсу, групи *СПРм-20-1*
Мороз М.Ю.
(прізвище, ініціали)
Спеціальність *122 – Комп'ютерні науки*
(код і повна назва спеціальності)
Тип програми *освітньо-професійна*
(освітньо-професійна або освітньо-наукова)
Освітня програма *Системне проектування*
(повна назва освітньої програми)
Керівник д.т.н., *проф. Мінухін С.В.*
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри системотехніки _____ *Гребеннік І.В.*
(підпис) (прізвище, ініціали)

2021 р.

Харківський національний університет радіоелектроніки

Факультет *Комп'ютерних наук*
(повна назва)

Кафедра *Системотехніки*
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність *122 – Комп'ютерні науки*
(код і повна назва)

Тип програми *освітньо-професійна*
(освітньо-професійна або освітньо-наукова)

Освітня програма *Системне проектування*
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

«_____»_____
2021 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові *Морозу Михайлу Юрійовичу*
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження процесів контролю якості взаємодії користувачів з веб-додатком інтернет-магазину
затверджена наказом по університету від «08» листопада 2021 р. № 1664СТ
2. Термін подання студентом роботи до екзаменаційної комісії 15 грудня 2021р
3. Вихідні дані до роботи: Перелік використовуваних програмних засобів: ОС Windows 10, IDE Visual Studio 2019, браузер Google Chrome, Edge, Firefox. Технічне забезпечення: комп'ютер зі встановленим Visual studio.
4. Перелік питань, що потрібно опрацювати в роботі 4. Зміст пояснювальної записки (перелік питань, що потрібно розробити) 4.1 Вступ. 4.2 Аналіз предметної області. 4.3 Дослідження засобів для автоматизованого тестування UI веб-додатку. 4.4 Розробка фреймворку автоматизації UI веб-додатку. 4.5 Висновки.

5.Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 5.1 Життєвий цикл розробки ПЗ 5.2 Модель якості ПЗ 5.3 Процес забезпечення якості 5.4 Структура веб-додатку 5.5 Компоненти Selenium 5.6 Компоненти Selenium WebDriver 5.7 Робочий процес Selenium WebDriver 5.8 Типи локаторів 5.9 Взаємодія SpecFlow та BDD з веб-додатком 5.10 Тестовий сценарій реєстрації у веб-додаток 5.11 Згенерований клас визначених кроків з шаблонами методів 5.12 Локатори елементів реєстрації 5.13 Методи класу AuthenticationPage 5.14 Реалізація методів визначення кроків класу AuthenticationSteps 5.15 Діаграма структури та взаємодії компонентів 5.16 Реалізація класу DataManager 5.17 Реалізація класу WaitHelper 5.18 Реалізація класу CommonPageMap 5.19 Реалізація класу CommonPageAssertions 5.20 Реалізація класу DriverManager 5.21 Реалізація класу DriverFactory 5.22 Реалізація класу OptionsFactory

6. Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз завдання та пошук літератури за темою атестаційної роботи	08.11.2021	
2	Дослідження існуючих засобів	16.11.2021	
3	Постановка задачі	20.11.2021	
4	Розробка фреймворку	20.11.2021	
5	Оформлення пояснювальної записки та презентації	06.12.2021	
6	Подання атестаційної роботи до екзаменаційної комісії	14.12.2021	

Дата видачі завдання 08 листопада 2021 р.

Студент _____ Мороз М.Ю.
(підпис) (прізвище, ініціали)

Керівник роботи _____ д.т.н., проф. Мінухін С.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка до магістерської кваліфікаційної роботи : 112 стор., 30 рис., 1 табл, 20 джерел.

ВЕБ-ДОДАТОК, СТАНДАРТИ ЯКОСТІ, ФУНКЦІОНАЛЬНЕ ТЕСТУВАННЯ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ЗАБЕЗПЕЧЕННЯ ЯКОСТІ, ФРЕЙМВОРК АВТОМАТИЗАЦІЇ КОРИСТУВАЦЬКОГО ІНТЕРФЕЙСУ, СЕЛЕНІУМ, СЕЛЕНІУМ ВЕБ-ДРАЙВЕР.

Об'єкт дослідження – процеси контролю якості взаємодії користувачів з веб-додатком інтернет-магазину.

Предмет дослідження – засоби та методи взаємодії користувачів з веб-додатком інтернет-магазину.

Мета дослідження – дослідити та розробити фреймворк для автоматизації взаємодії користувача з користувацьким інтерфейсом веб-додатку (UI Automation Framework).

Результатом кваліфікаційної роботи є розроблений фреймворк автоматизованого тестування користувацького інтерфейсу.

ABSTRACT

Master’s Thesis: 112 p., 30 pic., 1 table, 20 source. Graphic material attestation work contains 22 poster.

WEB APP, QUALITY STANDARDS, FUNCTIONAL TESTING, SOFTWARE, QUALITY ASSURANCE, UI AUTOMATION FRAMEWORK, SELENIUM, SELENIUM WEBDRIVER.

The object of study - the processes of quality control of user interaction with the web application of the online store.

The subject of research - methods and tools of user interaction with the web application of the online store.

The purpose of the study is to research and develop a framework for automating the user interface in a web application.

The result of the qualification work is a developed framework for automated user interface testing.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 8

ВСТУП 9

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 10

1.1 Опис предметної області 10

1.1.1 Програмне забезпечення та Життєвий цикл розробки ПЗ 10

1.1.2 Стандарти якості ПЗ та рівні тестування 15

1.1.2.1 Характеристики якості ПЗ 15

1.1.2.2 Приймальне тестування 17

1.1.2.3 Системне тестування 18

1.1.2.4 Інтеграційне тестування 18

1.1.2.5 Модульне тестування 19

1.1.3 Моделі у SDLC 20

1.1.3.1 Agile модель 20

1.1.3.2 Водоспадна (Waterfall) модель 21

1.1.4 Забезпечення якості 21

1.1.5 Процес забезпечення якості веб-додатків 23

1.1.6 Види інтернет-додатків 25

1.1.6.1 eCommerce платформи 26

1.1.6.2 Software as a service платформи 26

1.1.6.3 B2B-портали 26

1.1.6.4 CRM 27

1.1.7 Основні складові веб-додатку 28

1.2 Постановка задачі 30
2 ДОСЛІДЖЕННЯ ЗАСОБІВ ДЛЯ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ UI ВЕБ-ДОДАТКУ 31
2.1 Selenium 31
2.2 Selenium WebDriver 33
2.3 Локатори Selenium 36
2.3.1 Види локаторів 38
2.4 SpecFlow 40
2.5 Засоби модульного тестування веб-додатків 42
3 РОЗРОБКА ФРЕЙМВОРКУ АВТОМАТИЗАЦІЇ UI ВЕБ-ДОДАТКУ 43
3.1 Структура та компоненти 43
3.2 Діаграма фреймворку та реалізація класів 51
ВИСНОВКИ 60
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ 61
Додаток А 63
Додаток Б 88
Додаток В 110

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних;

СКБД – система керування базами даних;

ПЗ – програмне забезпечення;

ОС – операційна система;

QA – забезпечення якості;

SDLC – життєвий цикл розробки програмного забезпечення;

HTTP – протокол передачі гіпертексту;

SRS – специфікація вимог програмного забезпечення;

HTML – мова гіпертекстової розмітки;

IDE – інтегроване середовище розробки;

API – програмний інтерфейс програми;

DOM – об'єктна модель документа;

GUI – графічний інтерфейс користувача;

TCP – протокол керування передаванням;

IP – інтернет протокол;

RC – віддалений доступ;

CI/CD – безперервна інтеграція та безперервне розгортання;

ВСТУП

За останні роки використання інтернету та комп’ютерів значно зросло. Це зростання призвело до збільшення мережі, яка використовується через Інтернет, відома як всесвітня павутина.

Натепер підприємства з усіх сфер покладаються на веб-додатки для виконання своєї повсякденної діяльності. Ця популярність обумовлена такими властивостями, як доступність з любої точки світу та відносна незалежність від програмно-апаратної платформи, використовуваної кінцевими користувачами. Такий зріст значно підвищує значимість проблеми забезпечення якості, тому стало важливим, що дані, які мають ці підприємства були безпечними, без критичних помилок, повністю функціональними і працювали 24 години на добу, 7 днів на тиждень та 365 днів на рік. Для забезпечення цього процесу веб-додатки повинні пройти процес відомий як забезпечення якості. Процеси забезпечення якості повинні застосовуватися до кожного веб-додатка, щоб він залишався повністю функціональним навіть у критичний час.

У сучасному світі важно вирішити цю проблему, оскільки, на відміну від програмного забезпечення, веб-додатки повинні змінюватися з часом, щоб задовольнити потреби клієнтів. Під час розробки необхідно подбати не тільки про вимоги клієнтів, але й про зміну платформ. Інша проблема полягає у тому, що для веб-розробки зазвичай дається менше часу розробникам в порівнянні з настільним програмним забезпеченням.

Тому існують стандарти якості, які можуть допомогти зрозуміти що наше програмне забезпечення відповідає стандартам та може бути використаним по призначенню.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

1.1.1 Програмне забезпечення та Життєвий цикл розробки ПЗ

Програмне забезпечення (ПЗ) – це набір інструкцій, даних або програм, які використовуються для роботи на комп’ютері та виконанні конкретних завдань. Простіше кажучи, програмне забезпечення інструктує комп’ютеру, як функціонувати. Це загальний термін, який використовується для позначення сценаріїв і програм, які запускаються на таких пристроях, як настільні комп’ютери, мобільні телефони, планшети та інші розумні пристрої.

Без програмного забезпечення більшість комп’ютерів були б марними. Наприклад, веб-браузер — це програма, яка дозволяє користувачам отримати доступ до Інтернету. Без програмного забезпечення веб-браузера перегляд різноманітних сторінок веб-сайтів було б неможливим. Операційна система (ОС) являє собою програмне забезпечення, яке служить у якості інтерфейсу між додатками і апаратними засобами на комп'ютері або мобільному пристрої. TCP/IP вбудований у всі основні операційні системи, дає можливість комп’ютерам спілкуватися в мережах на великій відстані. Без ОС або вбудованих в неї протоколів було б неможливо отримати доступ до веб-браузера.

Більшість програмного забезпечення написані мовами програмування високого рівня, оскільки мова ближча до природної людської мови, а не до машинної. Потім мова високого рівня перекладається в машинний код низького рівня за допомогою компілятора або інтерпретатора для розуміння комп’ютером.

Життєвий цикл розробки програмного забезпечення (SDLC) — це серія кроків, які має виконати команда розробників програмного забезпечення для розробки та підтримки програмного забезпечення [1].

Життєвий цикл розробки програмного забезпечення починається з прийняття рішення про створення ПЗ і закінчується після видалення цього програмного забезпечення з експлуатації. Процес розробки ПЗ включає 5 основних етапів.

Кожен з них містить кілька кроків. По суті, SDLC — це дорожня карта для розробки програмного продукту.

До ключових переваг SDLC слідує віднести:

1. Забезпечує видимість для сторін, які займаються розробкою програмного забезпечення;
2. Дає можливість власникам бізнесу зберегти контроль над проектом;
3. Забезпечує передбачувані поставки протягом усього процесу розробки програмного забезпечення;
4. Мінімізує ризик, наприклад, перевищення бюджету або терміну розробки;
5. Гарантує, що процес розробки програмного забезпечення продовжуватиметься до тих пір, поки усі очікування не будуть виправдані.

Таким чином, усі проекти повинні мати окреслений життєвий цикл розробки програмного забезпечення, оскільки це єдиний спосіб гарантувати, що отримане програмне забезпечення відповідатиме вимогам як власників бізнесу, так і кінцевих користувачів. На рис. 1.1 наведено Життєвий цикл розробки ПЗ [1].

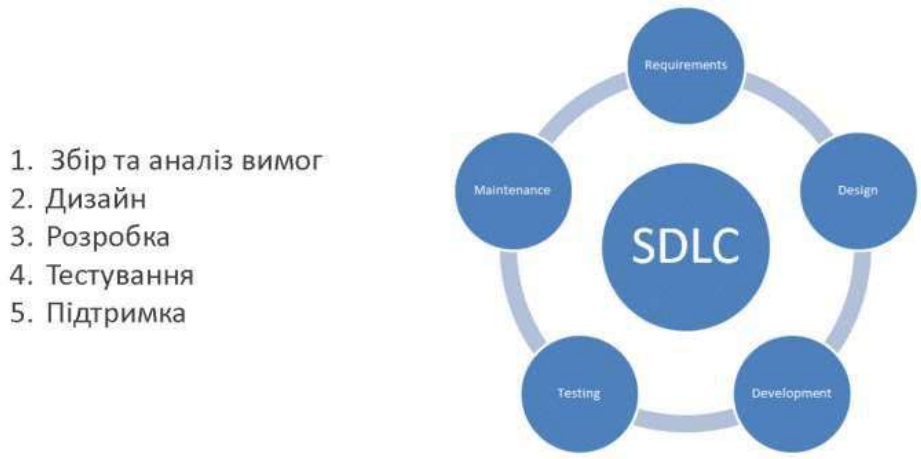


Рисунок 1.1 - Життєвий цикл розробки ПЗ.

1. Перший етап – Збір та аналіз вимог.

Мета цього кроку - зібрати та задокументувати бізнес-вимоги.

Цей крок включає збір вимог до розробки програмного продукту від зацікавлених сторін, експертів галузі та навіть потенційних клієнтів. Після цього власники проекту окреслюють масштаб проекту, визначаючи бюджет, ресурси, терміни, а також потенційні ризики та вимоги до забезпечення якості.

Після завершення збору вимог проводиться аналіз для перевірки доцільності розробки продукту. У разі виникнення будь-якої неясності проводиться дзвінок для подальшого обговорення.

Після чіткого розуміння вимоги створюється документ SRS (Software Requirement Specification). Цей документ повинен бути добре зрозумілий розробниками, а також повинен бути переглянутий замовником для подальшого використання [1].

2. Другий етап – Проектування.

Мета цього кроку – визначити архітектуру кінцевого складу програмного продукту з вимогами розробки програмного забезпечення.

Цей етап життєвого циклу розробки програмного забезпечення передбачає проектування всієї системи та її елементів, включаючи проектування високого рівня та проектування низького рівня. Проектування високого рівня визначається як архітектурний проект системи, тоді як проектування низького рівня — це проектування її компонентів. Таким чином, проектування низького рівня – це детальний опис усіх компонентів, конфігурацій та процесів IT-інфраструктури.

Затверджений дизайн системи визначає перелік програмних компонентів, які необхідно розробити, взаємодію з третіми сторонами, функціональні характеристики програми, бази даних, які необхідно використовувати та багато іншого. Дизайн, як правило, фіксується окремим документом – дизайн специфікацією [1].

На цьому етапі для спрощення процесу візуалізації використовуються так звані нотації – схематичні зображення системи, що розробляється. Основі нотації:

– Блок-схеми;

– ER-діаграми;

– UML-діаграми;

– Макети – наприклад, намальований у фото-редакторі прототип сайту [1].

3. Третій етап – розробка ПЗ

Мета цього кроку - створити справжнє програмне забезпечення.

Після того як вимоги і дизайн продукту затверджено, відбувається перехід до наступної стадії життєвого циклу – безпосередньої розробки. Тут починається написання програмістами коду програми відповідно до визначених раніше вимог.

Окрім того, програмісти пишуть модульні-тести для перевірки правильності роботи коду кожного компонента системи, проводять перевірку написаного коду, створюють збірки і розгортають готове ПЗ в програмному середовищі. Цей цикл повторюється до тих пір, поки всі вимоги не будуть реалізовані. Програмування передбачає чотири основні стадії [1]:

- розробка алгоритмів – фактично, створення логіки роботи програми;
- написання коду;
- компіляція – перетворення в машинний код;
- тестування та відлагодження – мова йде, головним чином, про модульне тестування.

4. Четвертий етап – тестування.

Мета цього кроку - забезпечити відповідність програмного забезпечення вимогам.

Після того, як команда розробників завершила програмування програмного забезпечення, настав час втрутитися інженерам з забезпечення якості (QA). Команда QA тестує програмне забезпечення, щоб виміряти його якість. На цьому етапі програмне забезпечення проходить різні види тестів таких як:

- функціональне тестування: забезпечення відповідності програмного забезпечення вимогам, описаним у специфікації вимог до програмного забезпечення;
- тестування продуктивності: спрямоване на визначення того, як працює програмне забезпечення під робочим навантаженням (швидкість реагування та стабільність);
- модульне тестування: тестування кожного компонента окремо. Якщо в будь-якому з них є недолік, розробники програмного забезпечення, відповідальні за це, повинні повернутися і виправити його.
- тестування безпеки: як випливає з назви, цей тип тестування спрямований на перевірку безпеки системи;

Розробники програмного забезпечення виправляють будь-які помилки, які виникають на цьому етапі, а потім команда QA знову тестує програмне забезпечення або його компонент. Тестування повторюється до тих пір, поки не будуть досягнуті критерії його закінчення. Забезпечення якості – це постійний процес, який триває до тих пір, поки програмне забезпечення не буде позбавлене від критичних помилок і не буде відповідати вимогам [1].

5. П’ятий етап – розгортання

Мета цього кроку - надати користувачам готове програмне забезпечення. Перевірена версія програмного забезпечення поставляється на ринок для бета-тестування. Команда підтримки збирає відгуки від перших користувачів, і якщо на цьому етапі виявляються помилки, розробники програмного забезпечення виправляють їх. Після цього виходить нова, покращена версія. Етап розгортання також включає подальше обслуговування програмного забезпечення та його постійне посилення [1].

1.1.2 Стандарти якості ПЗ та рівні тестування

Існує набір стандартів ISO 9000, який регулює загальні принципи забезпечення якості у всіх галузях. Найбільш важливими стандартами в розробці ПЗ є:

- ISO 9000:2000 - Системи управління якістю — Основи та терміни;
- ISO 9001:2000 - Системи управління якістю — Вимоги. Моделі для забезпечення якості проектування, розробки, виробництва, встановлення та обслуговування;
- ISO 9004:2000 Системи управління якістю — Рекомендації щодо підвищення ефективності;
- ISO/IEC 90003:2004 Інженерія програмного забезпечення — Керівництво щодо застосування ISO 9001:2000 до комп'ютерного програмного забезпечення [2].

IEEE 1008-1987 - Стандарт IEEE для модульного тестування програмного забезпечення. Для визначення повноти тестування він використовує інформацію про проект і реалізацію блоку, на додаток вимог до блоку. Описаний процес тестування складається з ієрархії етапів, заходів і завдань та визначає мінімальний набір завдань для кожного виду діяльності. Стандарт може застосовуватися до модульного тестування будь-якого програмного або мікро-програмного забезпечення цифрового комп'ютера, а також до тестування як нещодавно розроблених, так і модифікованих блоків. Концепції розробки програмного забезпечення та припущення щодо тестування, на яких базується цей стандартний підхід [2].

1.1.2.1 Характеристики якості ПЗ

Функціональність – визначається здатністю ПЗ вирішувати завдання, які відповідають зафіксованим і очікуваним потребам користувача, за заданих умов використання ПЗ. Тобто ця характеристика відповідає за те, що ПЗ працює справно і точно, функціонально сумісно, відповідає стандартам галузі та захищене від несанкціонованого доступу.

Надійність – здатність ПЗ виконувати необхідні завдання у зазначених умовах протягом заданого проміжку часу або вказаної кількості операцій. Атрибути даної характеристики – це завершеність і цілісність всієї системи, здатність самостійно і правильно відновлюватися після збоїв у роботі, відмовостійкість.

Зручність використання – можливість легкого розуміння, вивчення, використання і привабливість ПЗ для користувача.

Ефективність – здатність ПЗ забезпечувати необхідний рівень продуктивності згідно з виділеними ресурсами, часом та іншими зазначеними умовами.

Зручність супроводу – легкість, з якою ПЗ може аналізуватися, тестуватися, змінюватися для виправлення дефектів, реалізації нових вимог, для полегшення подальшого обслуговування та адаптуватися до наявного оточення.

Портативність – характеризує ПЗ з точки зору легкості його перенесення з одного оточення (software/hardware) в інше.

На даний момент найбільш поширена і найширше використовується багаторівнева модель якості програмного забезпечення, представлена у наборі стандартів ISO 9126. На верхньому рівні виділено 6 основних характеристик якості ПЗ, кожну з яких визначають набором атрибутів з відповідними метриками для подальшої оцінки на рис. 1.2 [3].

Рисунок 1.2 - Модель якості програмного забезпечення (ISO 9126-1)

1.1.2.2 Приймальне тестування

Приймальне тестування – вид тестування, що проводиться на етапі здачі готового продукту (або готової частини продукту) замовнику. Метою такого тестування є визначення готовності продукту, що досягається шляхом проходу тестових сценаріїв та випадків, які побудовані на основі специфікації вимог до ПЗ, що розробляється.

Результатом приймального тестування може стати:

- Відправка проекту на доопрацювання.
- Ухвалення його замовником, в якості виконаної задачі.

Це фінальний етап тестування продукту перед його релізом. При цьому, він не є дуже ретельним, всеохоплюючим та повним – тестується, зазвичай, тільки основний функціонал [4].

1.1.2.3 Системне тестування

Системне тестування - це тестування програмного забезпечення, що виконується на повній, інтегрованій системі, з метою перевірки відповідності системи вихідним вимогам, як функціональним, так і не функціональним.

Виконуючи системне тестування, можна виявити наступні типи дефектів:

- Неправильне використання системних ресурсів.
- Непередбачені комбінації даних користувача.
- Проблеми з сумісністю оточення.
- Непередбачені сценарії використання.
- Невідповідність функціональним вимогами.
- Погана зручність використання.

Можна виділити 2 підходи до системного тестування:

На базі вимог. Тестування проводиться відповідно до функціональних або нефункціональних вимог, для кожного з яких пишеться тестовий сценарій.

На базі випадків використання. Тестування відбувається відповідно до варіантів використання продукту, на основі яких створюються сценарії використання. Для кожного сценарію використання створюються свої тестові сценарії [5].

1.1.2.4 Інтеграційне тестування

Інтеграційне тестування – вид тестування, при якому на відповідність вимог перевіряється інтеграція модулів, їх взаємодія між собою, а також інтеграція підсистем в одну загальну систему. Для інтеграційного тестування використовуються компоненти, вже перевірені за допомогою модульного тестування, які групуються у множини. Дані множини перевіряються відповідно до плану тестування, складеним для них, а об’єднуються вони через свої інтерфейси.

Існує декілька підходів до інтеграційного тестування:

Підхід низу вгору. Спочатку збираються і тестуються модулі найнижчих рівнів, а потім по зростанню до вершини ієрархії. Даний підхід вимагає готовності усіх зібраних модулів на всіх рівнях системи.

Підхід зверху вниз. Даний підхід передбачає рух з модулів високого рівня вниз. При цьому використовуються заглушки для тих модулів, які знаходяться нижче за рівнем, але включення яких до тесту ще не відбулося.

Великий вибух. Всі модулі всіх рівнів збираються разом, а потім тестується. Даний метод економить час, але вимагає ретельного опрацювання тест кейсів [6].

1.1.2.5 Модульне тестування

Модульне тестування - тестування кожної атомарної функціональності додатку окремо, у штучно створеному середовищі. Саме потреба у створенні штучного робочого середовища для певного модуля, вимагає від тестувальника знань в автоматизації тестування програмного забезпечення, деяких навичок програмування. Дане середовище для деякого модуля створюється за допомогою драйверів і заглушок.

Драйвер – визначений модуль тесту, який виконує елемент, що ми тестуємо. Заглушка – частина програми, яка симулює обмін даними із компонентом, що тестується, виконує імітацію робочої системи.

Перевага модульного тестування:

- Модульне тестування мотивує програмістів писати код максимально оптимізованим, проводити рефакторинг (спрощення коду програми, не зачіпаючи її функціональність), так як за допомогою модульного тестування можна легко перевірити працездатність розглянутого компонента.

– Необхідність відділення реалізації від інтерфейсу (зважаючи на особливості модульного тестування), що дозволяє мінімізувати залежності в системі.

– Документація модульних тестів може служити прикладом «живого документа»

Для кожного класу, який тестується даним способом, модульне тестування допомагає краще зрозуміти роль кожного класу на тлі всієї програмної системи.

– Також, при «розробці через тестування», яке активно використовується в екстремальному програмуванні, модульне тестування є одним з основних інструментів, що дозволяє розробляти продукт відповідно до вимог до даного модулю [7].

1.1.3 Моделі у SDLC

1.1.3.1 Agile модель

Завдяки підходу Agile розробники програмного забезпечення можуть швидко адаптуватися до ринкової ситуації, оскільки ця модель дозволяє вносити зміни в продукт на будь-якому етапі процесу розробки програмного забезпечення. Цей підхід ідеально підходить для проектів із різними вимогами.

Цей метод дозволяє створювати продукти за допомогою коротких циклів «спринтів», де кожен спринт закінчується робочим продуктом з обмеженою кількістю функцій. Кожен спринт включає проектування, розробку, тестування та розгортання.

Перевага цього підходу полягає в тому, що власники продуктів можуть бачити результати кожного короткого циклу, надавати свої відгуки та вносити корективи, якщо це необхідно. На початку наступного циклу розробники програмного забезпечення переглядають попередню версію продукту та представляють її для наступного раунду зворотного зв’язку. Таким чином, життєвий цикл розробки програмного забезпечення Agile відомий як безперервний процес [8].

Основні характеристики Agile наступні:

- показ результатів кожен спринт, можливість додавання нових функцій на ходу;
- тестування проводиться протягом усієї розробки програмного забезпечення;
- постійне спілкування між замовниками, розробниками програмного забезпечення, проектом, що призводить до покращення версій після кожного спринту;
- забезпечення якості є ключовим процесом.

1.1.3.2 Водоспадна (Waterfall) модель

Оскільки Agile і Waterfall це різні підходи до розробки програмного забезпечення, вони підходять для різних типів проектів. Модель Waterfall є хорошим рішенням для проектів зі стабільними та визначеними вимогами, тоді як Agile найкраще підходить для проектів із різними та не повними вимогами.

Waterfall сприяє жорсткому підходу до розробки програмного забезпечення в порівнянні з гнучким підходом Agile. Ця модель не передбачає впровадження будь-яких змін у процесі розробки програмного забезпечення. Розробники програмного забезпечення можуть переходити до наступного етапу лише після завершення попереднього. Таким чином, буде лише одна версія програмного забезпечення, тоді як у Agile кожен спринт призводить до робочої версії програмного забезпечення [9].

Основні характеристики моделі Waterfall включають:

- жорстка послідовність кроків розробки;
- перехід до наступної фази можливий лише після завершення попередньої фази;
- фіксований бюджет;
- замовники не залучені до процесу розробки програмного забезпечення;
- зміни можуть бути реалізовані лише після завершення процесу розробки.

1.1.4 Забезпечення якості

Якість програмного забезпечення - це сукупність параметрів програмного забезпечення, які стосуються його здатності задовольняти встановлені і передбачувані потреби[10].

Забезпечення якості - це сукупність заходів, що охоплюють всі технологічні етапи розробки, випуску та експлуатації програмного забезпечення інформаційних систем, що вживаються на різних стадіях життєвого циклу розробки ПЗ, для забезпечення необхідного рівня якості продукту, що випускається [10].

Тестування програмного забезпечення - це одна з технік контролю якості, що включає в себе активності з планування робіт, проектування тестів, виконання тестування та аналізу отриманих результатів [10].

Верифікація - це процес оцінки системи або її компонентів з метою визначення, чи задовольняють результати поточного етапу розробки вимогам, сформованих на початку цього етапу. Тобто, чи виконуються наші цілі, терміни, завдання розробки проекту, визначених на початку поточної фази.

Валідація - це відповідність розроблюваного ПЗ очікуванням та потребам користувача, вимогам до системи.

Методологія забезпечення якості має визначений цикл, який називається циклом PDCA або циклом Демінга проілюстрований на рис. 1.3. Фазами цього циклу є Plan, Do, Check, Act [11].

Рисунок 1.3 - Процес забезпечення якості

Ці кроки повторюються, щоб гарантувати, що процеси, які дотримуються в організації, оцінюються та вдосконалюються на періодичній основі. Давайте розглянемо вищезазначені етапи процесу забезпечення якості:

Plan: організація повинна планувати та встановлювати цілі, пов’язані з процесом, і визначити процеси, які необхідні для отримання високоякісного кінцевого продукту.

Do: розробка та тестування процесів, а також «внесення» змін у процеси.

Check: моніторинг процесів, змінення процесів і перевірка, чи відповідає він поставленим цілям.

Act: тестер із забезпечення якості повинен виконувати дії, необхідні для покращення процесів.

Організація повинна використовувати гарантію якості, щоб продукт був розроблений та впроваджений з правильними процедурами. Це допомагає зменшити проблеми та помилки у кінцевому продукті.

1.1.5 Процес забезпечення якості веб-додатків

Забезпечення якості веб-додатків зводиться до тестування шести областей, які включають тестування функціональності, тестування зручності, тестування інтерфейсу, тестування сумісності, тестування продуктивності та тестування безпеки.

Тестування функціональності: проводиться щоб переконатися, що веб-додаток повністю виконує свої функції, у табл. 1.1 наведені області перевірки [12].

Таблиця 1.1 – Області перевірки під час функціонального тестування.

Посилання	Вихідні та внутрішні посилання, перевірка на пошкодження посилань
Форми	Перевірка валідації, значення по замовчуванню, редагування, видалення форми
Кукі	Поведінка додатку після видалення кукі, перевірка на шифрування кукі
База даних	Перевірити цілісність даних після оновлення, додавання, редагування, видалення

Тестування зручності використання. Зручність використання означає, наскільки легко користувач відчуває себе під час використання веб-програми. Щоб

перевірити зручність використання, необхідно перевірити навігацію, зміст та будь-яку іншу річ, яка може якось допомогти користувачеві. Темних кольорів слід уникати, оскільки вони ніяким чином не допомагають користувачеві. Зображення та інші нетекстові речі повинні бути правильно розміщені, щоб вони не відволікали користувача. «Contact us» та «Search» мають бути розміщені на видному місці, щоб користувач міг їх легко знайти [12].

Тестування інтерфейсу. Основними інтерфейсами у веб-додатку є веб-сервер і інтерфейс сервера додатків, а також інтерфейс сервера додатків і сервера баз даних. Слід перевірити, чи взаємодія між цими інтерфейсами виконується належним чином, а помилки (якщо такі є) обробляються правильно. Якщо виникає якась помилка, вона повинна бути відображена користувачеві [12].

Тестування на сумісність: Найважливішою особливістю будь-якого веб-додатка є сумісність, оскільки користувачі запускають веб-програму на різних платформах і в різних браузерах. Веб-додаток має бути ретельно перевірений у різних веб-браузерах та різних операційних системах. Багато веб-додатків мають мобільну версію, коли користувач отримує доступ до них з мобільного пристрою. Якщо під час доступу до веб-програми на мобільному пристрої виникають якісь проблеми, їх слід вирішити. Ще один аспект – друк. Слід переконатися, що під час друку у веб-програмі, текст, зображення та інші елементи повинні правильно вирівнюватися та підходити до розміру сторінки, вибраного користувачем перед друком [12].

Тестування продуктивності. Тестування продуктивності виконується для того, щоб перевірити, як веб-додаток (або система) буде вести себе за різних обставин, наприклад, під час нормального та високого використання. Для цього потрібно провести навантажувальне тестування та стрес-тестування. Під час навантажувального тестування потрібно протестувати веб-додаток, коли до веб-програми одночасно звертається велика кількість користувачів. Треба перевірити, чи працює веб-додаток у час пік - може веб-додаток обробляти великі запити користувачів, одночасні запити доступу до бази даних, тощо. Під час стрес-тестування система перевіряється за межі її можливостей. Мета стрес-тестування – побачити, як система поводиться під час великого навантаження та як вона відновлюється після збоїв [12].

Тестування безпеки: Для перевірки безпеки веб-додатку, потрібно перевірити конкретні URL-адреси, які не повинні бути доступні для певного користувача. Наприклад, при перегляді документу, URL-адреса якого в кінці має ID=123. Якщо документ з ID=124 недоступний, то при зміні URL-адреси має відображатися відповідне повідомлення про помилку. Для запобігання автоматичного трафіку слід використовувати CAPTCHA [12].

1.1.6 Види інтернет-додатків

Все залежить від завдань, які необхідно вирішити. Тому попередньо проводиться оцінка бізнесу, його аналіз. Це дозволяє зрозуміти – що саме слід реалізувати, і яким чином.

Веб-додаток може бути найрізноманітнішим, все залежить від сфери, в якій саме він буде створений і запущений. Хоча такий поділ умовний. Тому що легко придумати безліч тематик, напрямків, для яких буде актуальна реалізація такого продукту, наприклад:

- бронювання готелів, квитків, автомобілів;
- інтернет-магазини;
- замовлення їжі, різних послуг;
- розваги, ігри;
- фінансові, банківські послуги різного типу – починаючи від калькулятора валют і закінчуючи повноцінним онлайн-банкінгом;
- соціальні мережі;
- освітні програми – наприклад, з вивчення іноземної мови;
- біржі фрілансу;
- CRM та багато іншого.

1.1.6.1 eCommerce платформи

До додатків електронної комерції відносять інтернет-магазини, маркетплейси і онлайн-аукціони. Для таких додатків важлива інтеграція з платіжними системами і службами доставки. Інші пріоритети включають створення комфортного користувацького досвіду та забезпечення безпеки. Електронна комерція охоплює цілий ряд різних видів бізнесу та корпорацій і стає одним з найважливіших аспектів Інтернету. На сьогоднішній день онлайн-торгівля становить понад 5% світового обсягу торгівлі. Власники великого, середнього і малого бізнесу залучаються до процесу електронної торгівлі і розуміють, що розробка торгових майданчиків і складних інтернет-магазинів займає перше місце у масштабуванні бізнесу.

1.1.6.2 Software as a service платформи

Software as a Service - це модель розповсюдження програмного забезпечення, при якій користувачі отримують доступ до програми через браузер або інший веб-інтерфейс. На відміну від класичного ПЗ, SaaS-додаток не потрібно встановлювати на комп'ютер - він працює відразу після завантаження в браузер.

Переваги цієї платформи в тому що використовувати SaaS програму можна на будь-якому пристрої, підключеному до інтернету: ПК, ноутбуку, смартфону. SaaS-додатки мають гнучкі можливості по інтеграції з іншими інтернет-сервісами через API. Корпоративні SaaS-рішення допомагають автоматизувати внутрішні бізнес-процеси на підприємствах. Веб-додатки можуть виступати центральною базою даних для всіх інших сервісів і додатків компанії.

1.1.6.3 B2B-портали

B2B портал - це інтернет-майданчик, в межах якого реалізуються угоди між компаніями (наприклад, виробником і гуртовиком, між гуртовиком і роздрібним продавцем). До порталу можна підключити партнерів з різних рівнів ланцюга розподілу: дистриб'юторів, дилерів, незалежних гуртовиків, ритейлерів.

На порталі здійснюються оптові продажі, оформлення і попередня обробка замовлень, обмін документами і довідковою інформацією. Постачальник може скоротити час на взаємодію з покупцем, розвантажити відділи продажів і підтримки. Замовник може знаходити потрібну продукцію і всю інформацію про неї в інтерактивному каталозі з фільтрами і сортуванням, моментально отримувати довідки без необхідності дзвінків постачальнику. То за рахунок того, що частина бізнес процесів пов'язаних з продажами переноситься в B2B портал, замовник дозволяє цьому інструменту, зробити хоча б частину тієї роботи, яку виконує щодня менеджер вручну. І таким чином, компанія вирішує питання не тільки з навантаженням на персонал, але і скорочує час на обробку замовлення у рази.

1.1.6.4 CRM

Такий веб-додаток призначений для спрощення взаємовідносин з клієнтами, покупцями. Він дозволяє ефективно вирішувати різні завдання, серед яких контроль і планування [13].

Також слід виділити наступні завдання:

- збереження бази клієнтів і швидкий доступ до неї;
- чітка аналітика за усіма замовленнями, продажами;
- збільшення обсягу продажів;
- оптимізація діяльності співробітників компанії;
- суттєве зниження використання паперових документів.

1.1.7 Основні складові веб-додатку

Веб-додаток – це клієнт серверний додаток, в якому клієнтом є браузер, а сервером – веб-сервер. На рис. 1.4 наведена структура веб-додатку.

Рисунок 1.4 – Структура веб-додатку

Веб-додаток складається з клієнта, серверу та бази даних. Зазвичай, клієнтом виступає браузер, але є випадки коли веб-сервер виконує запити до іншого, роль клієнта виконую перший сервер. Коли клієнтом є браузер, для того щоб користувач побачив графічний інтерфейс додатку у вікні браузера, останній повинен обробити отриману відповідь веб-сервера, де буде міститься інформація, яка реалізована із застосуванням HTML, CSS, JS. Веб-сервер – це сервер, який приймає та віддає HTTP-запити та відповіді. База даних у даному контексті це - інформаційна модель, котра дозволяє впорядковано зберігати дані про об’єкт чи групу об’єктів, які володіють набором властивостей, котрі можна категоризувати. База даних функціонує під керівництвом систем керування базами даних (СУБД). Серед них найбільш популярнішими СУБД є MySQL, MS SQL Server, PostgreSQL, Oracle. Дані веб-додатків зберігаються зазвичай на сервері, обмін інформацією відбувається по мережі [14].

Шлях запиту користувача починається з DNS:

1. Запит до DNS для отримання IP-адреси.
2. DNS віддає IP-адресу користувачеві.

3. Користувач звертається до веб-додатку по IP через TCP/IP протокол.
4. Відкривається мережеве з’єднання HTTP.

DNS або Domain Name System – це ієрархічна розподілена система перетворення імені хоста у IP-адресу. Усі хости, підключені до Інтернету, смартфони, настільні комп’ютери, сервери, знаходять один одного і обмінюються інформацією за допомогою цифр. Ці цифри називаються IP-адресами. Тому, щоб відкрити веб-сайт у браузері, можна написати доменне ім’я, наприклад google.com, а не запам’ятовувати довгі набори цифр [14].

Хост – це будь-який комп'ютерний пристрій, що має доступ до IP мережі. Стек протоколів TCP/IP – це набір мережевих протоколів, які забезпечують передачу даних. TCP – Transmission Control Protocol, IP – Internet Protocol.

1.2 Постановка задачі

Проаналізувавши предметну область, можна сказати, що існує велика кількість різноманітних інтернет-додатків, для більшості з яких, забезпечення якості зводиться до шести областей у які входять: функціональне тестування, тестування зручності використання, тестування користувацького інтерфейсу, тестування на сумісність, тестування продуктивності та тестування безпеки. Але тестування будь-якої з цих областей витрачає багато часу для перевірки веб-додатка вручну, а іноді навіть неможливим. Тому щоб уникнути більшості проблем при тестуванні веб-додатку, процеси потрібно автоматизувати, вбираючи фактор похибки людини. Для автоматизування процесу тестування потрібно розробити функціонуюче середовище у вигляді фреймворку, завдяки якому можна автоматизувати тестові сценарії для веб-додатку.

Таким чином можна виділити основні критерії для розробки фреймворку:

1. Запуск тестових сценаріїв у браузерах таких як: Google Chrome, Edge, Firefox.
2. Розробка методів які імітують дії користувача у веб-додатку.
3. Розробка архітектури фреймворку з можливістю перевикористання розроблених методів для уникнення дубляжу коду.

Опираючись на основні критерії для розробки фреймворку автоматизації тестування, в даній роботі ставляться такі завдання:

1. Провести дослідження існуючих засобів для автоматизованого тестування користувацького інтерфейсу веб-додатку.
2. Розробити необхідні компоненти для створення фреймворку.
3. Розробити та реалізувати методи для імітації дій користувача у веб-додатку.
4. Налаштувати запуск тестових сценаріїв у браузерах для основних критеріїв для розробки фреймворку.
5. Розробити фреймворк для автоматизації тестування взаємодії користувача з користувацьким інтерфейсом веб-додатку.

2 ДОСЛІДЖЕННЯ ЗАСОБІВ ДЛЯ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ UI ВЕБ-ДОДАТКУ

2.1 Selenium

Одним із чудових наборів інструментів автоматизації є Selenium, який надає можливості для автоматизації дій користувача у веб-додатку. Це загальний проект з відкритим кодом для ряду інструментів і бібліотек, які дозволяють та підтримують автоматизацію веб-браузерів [15].

Selenium — це набір інструментів веб-автоматизації з відкритим кодом, який використовує потужність веб-браузерів і допомагає автоматизувати робочі процеси взаємодії користувачів із веб-додатком у браузері. Кожен інструмент у наборі має конкретні унікальні можливості, які допомагають у проектуванні та розробці системи автоматизації. Усі ці компоненти можна використовувати як окремо, так і поєднувати один з одним для досягнення рівня автоматизації тестування. На рис. 2.1 показані різні компоненти Selenium Suite[15].

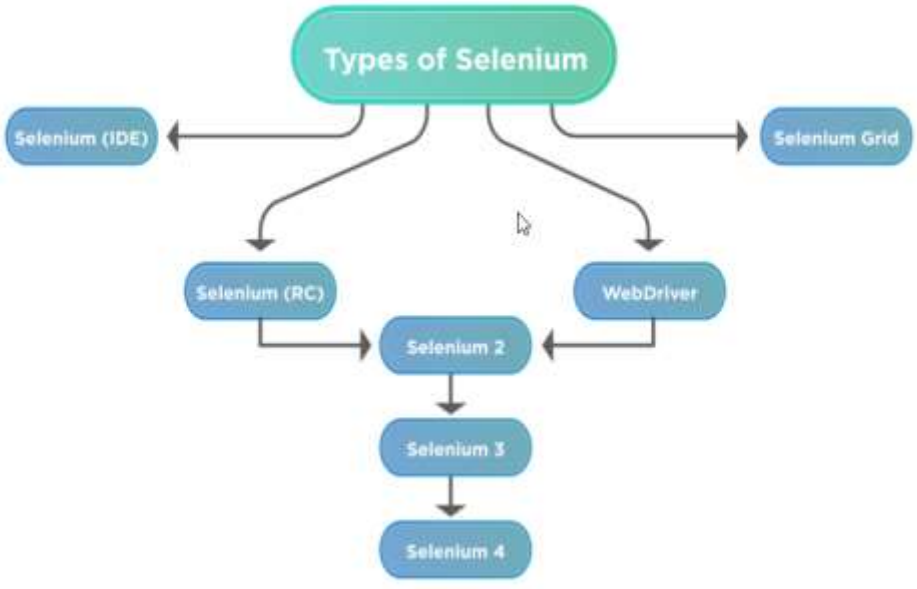


Рисунок 2.1 – Компоненти Selenium Suite

Selenium IDE — це розширення, доступне як для Firefox, так і для Chrome, яке має доступні функції запису та відтворення.

Selenium RC — це сервер, який виступає посередником між користувачем і браузером, який потребує взаємодії. RC використовує Javascript для роботи з браузерами, дозволяючи користувачам писати код мовою на свій вибір.

Selenium WebDriver є найбільш часто використовуваним компонентом Selenium. WebDriver дозволяє користувачам писати власний код мовою, яку вони вибирають, і взаємодіяти з браузером на свій вибір за допомогою драйверів для браузера. WebDriver працює на рівні ОС і використовує протокол під назвою JSONWireProtocol для зв'язку з браузерами [16].

Існує багато можливостей, які надає Selenium, що він є одним із найулюбленіших інструментів автоматизації на ринку. Далі наведені ці можливості [15]:

1. Відкритий вихідний код: Selenium є відкритим вихідним кодом, що означає, що для налаштування та використання Selenium не потрібно ніяких витрат. Selenium можна безкоштовно завантажити та використовувати.

2. Імітація дій користувача: майже всі реальні дії користувача, як натискання кнопок, перетягування та виділення, прапорці, натискання клавіш, натискання, прокрутка, можна автоматизувати за допомогою Selenium.

3. Легка реалізація: користувачі можуть розробляти власні розширення для свого використання, оскільки код є відкритим вихідним кодом.

4. Підтримка різних мов програмування: найважливішою перевагою Selenium є широка підтримка різних мов. Selenium підтримує такі мови програмування як: Java, Python, JavaScript, C#, Ruby, Perl, Haskell, Go та інші.

5. Підтримка браузера: Selenium може працювати з усіма постачальниками браузерів, які існують. Selenium підтримує Chrome, Firefox, Edge, Internet Explorer, Safari.

6. Підтримка ОС: прив'язки Selenium доступні для всіх основних ОС, таких як: Linux, macOS, Windows.

7. Підтримка Framework: Selenium підтримує кілька фреймворків, таких як Maven, TestNG, PYTest, NUnit, Mocha, Jasmine тощо. Selenium добре інтегрується з такими інструментами CI, як Jenkins, Circle CI, GOCD, Travis CI, Gitlab тощо.

8. Можливість повторного використання коду: скрипти, написані для Selenium, сумісні з різними браузерами.

2.2 Selenium WebDriver

Selenium Webdriver є одним із важливих членів цього сімейства і добре відомий своєю різноманітністю та стабільністю для веб-автоматизації. Selenium WebDriver — це набір API з відкритим кодом, який надає можливість взаємодії з будь-яким із сучасних веб-браузерів, а потім, у свою чергу, автоматизувати дії користувача з цим браузером. Це важливий компонент сімейства Selenium. Як відомо, Selenium не є самостійним інструментом; скоріше, це набір інструментів, які утворюють набір Selenium, який був створений під час об'єднання двох проектів Selenium RC і WebDriver. Далі будуть наведені функції, чому він є оптимальним вибором для веб-автоматизації [16].

1. Динамічна автоматизація веб-сторінок: може автоматизувати динамічні веб-сайти, на яких вміст сторінок змінюється в результаті дій користувача.

2. Працює близько до браузера: постачальники браузерів постачають свою реалізацію WebDriver. Тому вони тісно пов’язані з браузером, що забезпечує кращий досвід тестування.

3. Імітує реального користувача: Selenium WebDriver дозволяє QA інженерам імітувати типові дії користувачів на веб-сайтах.

4. Підтримує крос-браузерне тестування: Selenium WebDriver має найважливішу перевагу під час виконання крос-браузерного тестування — де QA інженер може тестувати той самий веб-сайт, використовуючи той самий фрагмент коду в різних браузерах. Він дозволяє перевіряти тестові випадки у кількох наборах браузерів одночасно.

5. Підтримує паралельне виконання: Якщо є більше сценаріїв для виконання у кількох браузерах, виконання їх один за одним займає багато часу. Таким чином, Selenium WebDriver дозволяє паралельне виконання, використовуючи такі фреймворки, як NUnit, щоб виконання тестових випадків було швидшим. Це дозволяє виконувати широкомасштабні тестові випадки за короткий час.

6. Підтримує перегляд результатів виконання: Selenium WebDriver дозволяє QA інженеру переглядати у реальному часі виконання автоматизованого тестового запуску на комп'ютерній системі, а також на будь-якому іншому хості конвеєра CI/CD, підтримуючи такі функції, як скріншот екрана, відеозапис тестових випадків тощо.

7. Підтримує сучасні методи розробки: Selenium WebDriver дуже добре інтегрується з сучасними принципами розробки програмного забезпечення, такими як Behavior Driven Development, завдяки інтеграції з бібліотекою Cucumber.

Будучи частиною загальної системи компонентів, можна зробити висновок, що Selenium WebDriver не є окремим інструментом тестування. Він містить різні компоненти, необхідні для виконання тестів. На рис. 2.2 наведені компоненти Selenium [15].

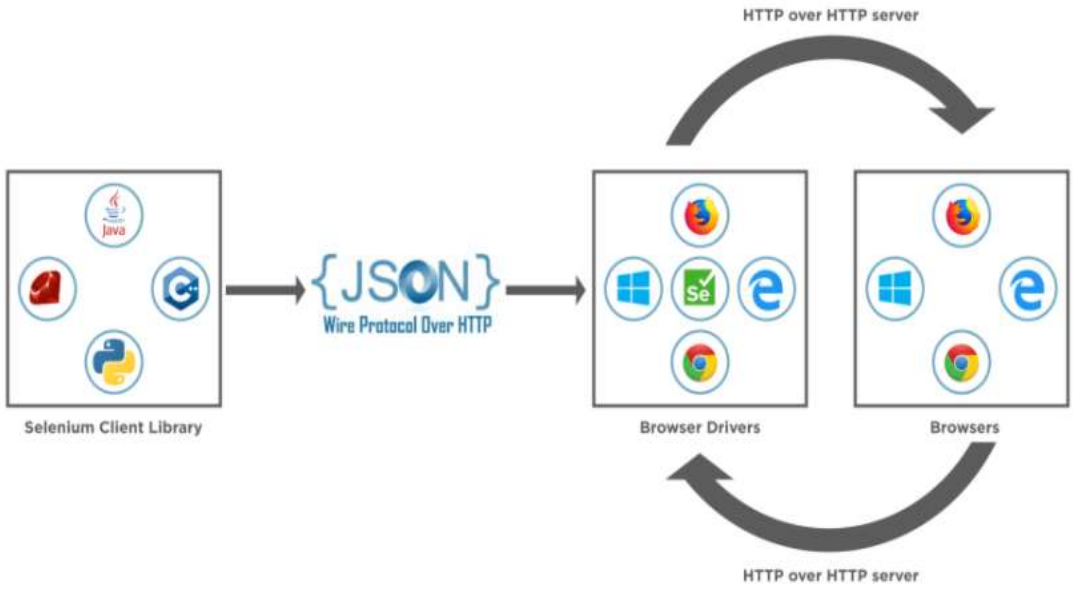


Рисунок 2.2 – Компоненти Selenium WebDriver

Протокол JSON Wire згідно з наведеними вище компонентами Selenium, полегшує всю комунікацію, яка відбувається в Selenium між браузером і кодом. Це серце Selenium. JSON Wire Protocol забезпечує середовище для передачі даних за допомогою API RESTful (Representational State Transfer), який забезпечує транспортний механізм і визначає веб-сервіс RESTful за допомогою JSON через HTTP [16].

Оскільки Selenium підтримує різні браузери, кожен браузер має власну реалізацію стандарту W3C, який надає Selenium. Тому доступні специфічні для браузера двійкові файли є специфічними для кожного браузера і приховують логіку реалізації від кінцевого користувача. Протокол JSON Wire встановлює з'єднання між двійковими файлами браузера і клієнтськими бібліотеками.

Selenium зможе запускати тести в браузерах, лише якщо вони встановлені локально, на локальному хості або на серверних хостах. Тому установка браузера необхідна. На рис. 2.3 наведено робочий процес Selenium WebDdiver [16].

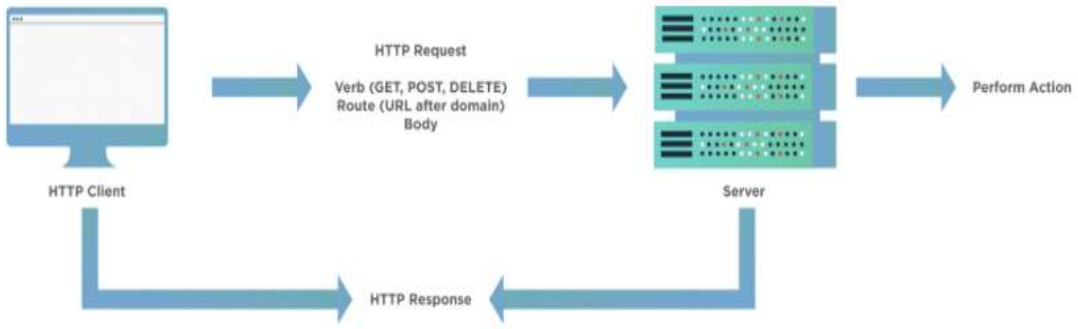


Рисунок 2.3 – Робочий процес Selenium WebDriver

Коли користувач пише код для WebDriver у Selenium і виконує його, у фоновому режимі відбуваються наступні дії [16]:

1. Генерується HTTP-запит, який надходить до відповідного драйвера браузера (Chrome, IE, Firefox). Для кожної команди Selenium існує індивідуальний запит. Драйвер браузера отримує запит через HTTP-сервер. HTTP-сервер вирішує,

які дії/інструкції потрібно виконати у браузері і тоді браузер виконує інструкції/кроки.

2. HTTP-сервер отримує статус виконання, а потім надсилає його у зворотному напрямку сценарію автоматизації, який потім формує результат.

2.3 Локатори Selenium

Локатори – це спосіб ідентифікації елемента HTML на веб-сторінці, і майже всі інструменти автоматизації інтерфейсу користувача надають можливість використовувати локатори для ідентифікації елементів HTML на веб-сторінці. Дотримуючись тієї ж тенденції, Selenium також має можливість використовувати локатори для ідентифікації елементів HTML. Selenium підтримує різні види локаторів. Визначення правильного елемента GUI на веб-сторінці є необхідною умовою для створення будь-якого успішного сценарію автоматизації [17].

Перше, з чого потрібно почати, — це знайти елемент HTML у DOM (об’єктна модель документа), для якого нам потрібно взяти локатор. Для визначення веб-елементу у DOM-дереві браузера виконаємо наступні кроки:

1. Доступ до DOM можна отримати в Google Chrome, натиснувши F12 або клацнувши правою кнопкою миші на веб-сторінці, а потім вибравши Inspect, що наведено на рис. 2.4 [18].

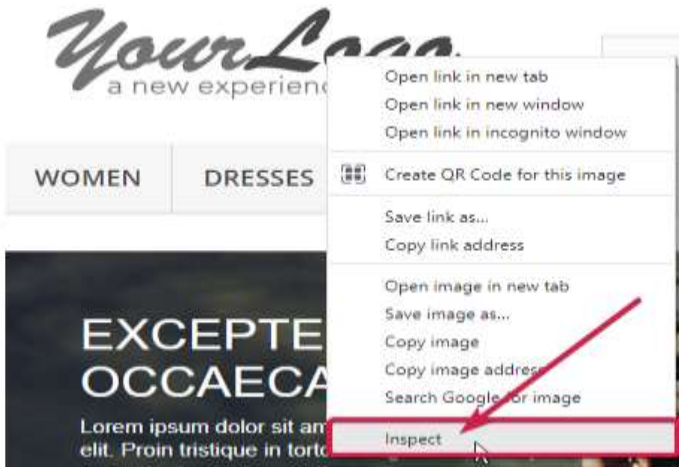


Рисунок 2.4 – Доступ до DOM-дерева веб-елементів

2. Після натискання опції «Inspect», відкриється консоль інструментів розробника, як показано на рис. 2.5. За замовчуванням він відкриє вкладку «Elements», яка представляє повну структуру DOM веб-сторінки. Тепер, якщо ми наведемо вказівник миші на теги HTML у DOM, він виділить відповідні елементи, які він представляє на веб-сторінці.

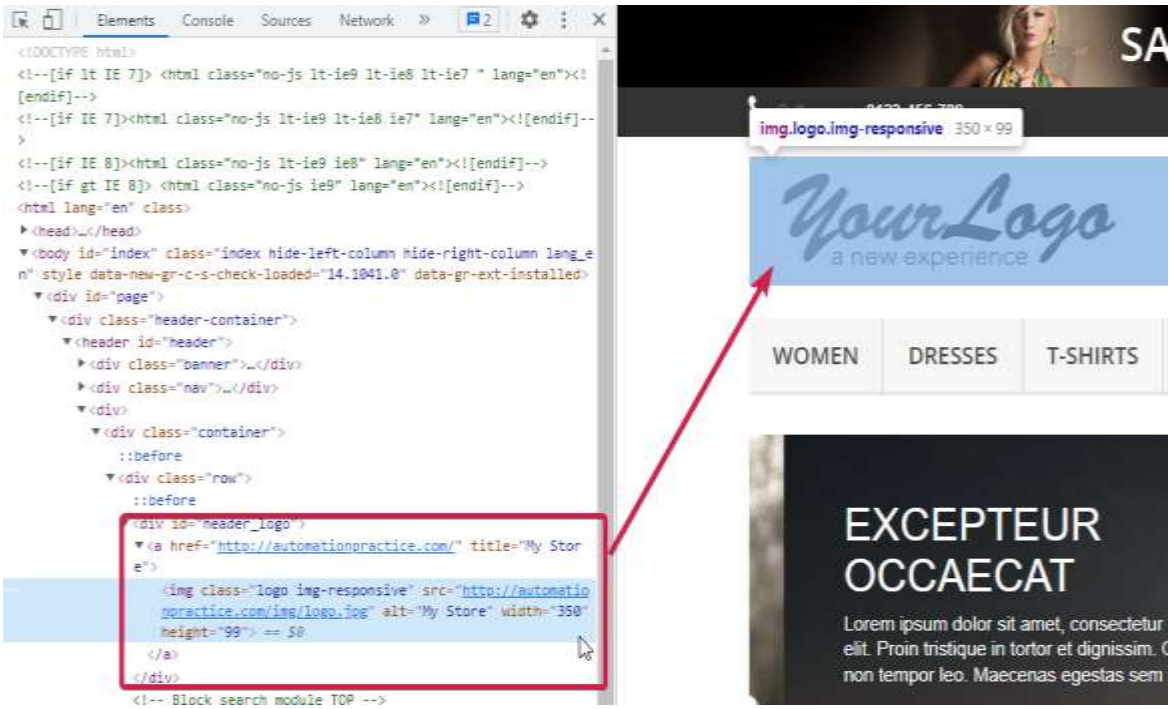


Рисунок 2.5 – Структура DOM дерева та виділений елемент

3. Для знаходження веб-елементу у DOM, потрібно натиснути стрілку «Значок миші» (маркер 1 на рис. 2.6), а потім вибрати веб-елемент на веб-сторінці. Він автоматично виділить відповідний елемент HTML у DOM. Наприклад, знайдемо елементи HTML, що відповідають зображенню логотипу (маркер 3). Наведемо вказівник миші та клацнемо зображення логотипу, й тоді автоматично виділиться відповідний елемент HTML (маркер 2):

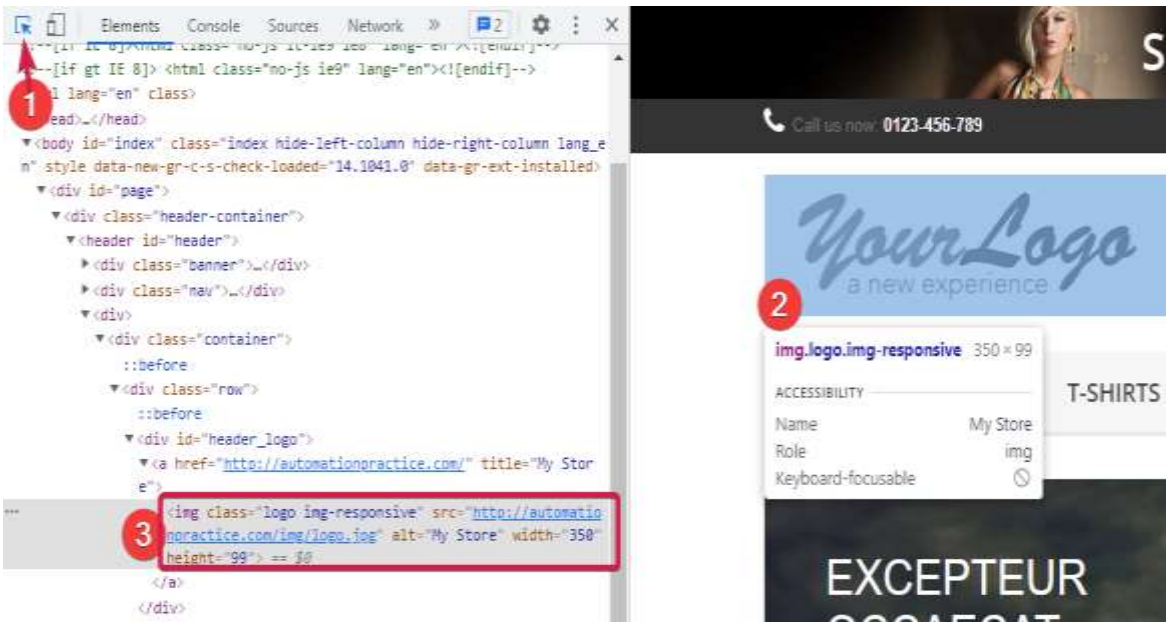


Рисунок 2.6 – Знайдений відповідний елемент HTML

Таким чином, можна легко знайти елемент HTML у DOM, що відповідає веб-елементу на веб-сторінці.

2.3.1 Види локаторів

Існують різні типи локаторів, за допомогою яких можна однозначно ідентифікувати веб-елемент на веб-сторінці. На рис. 2.7 показані типи локаторів, які підтримує Selenium.

Для доступу до всіх цих локаторів Selenium надає клас з іменем «By», який допомагає знайти елементи в DOM. Він пропонує кілька різних методів: className, cssSelector, id, linkText, name, partialLinkText, taName і xPath, які можуть ідентифікувати веб-елементи на основі їх відповідних стратегій локатора [17].

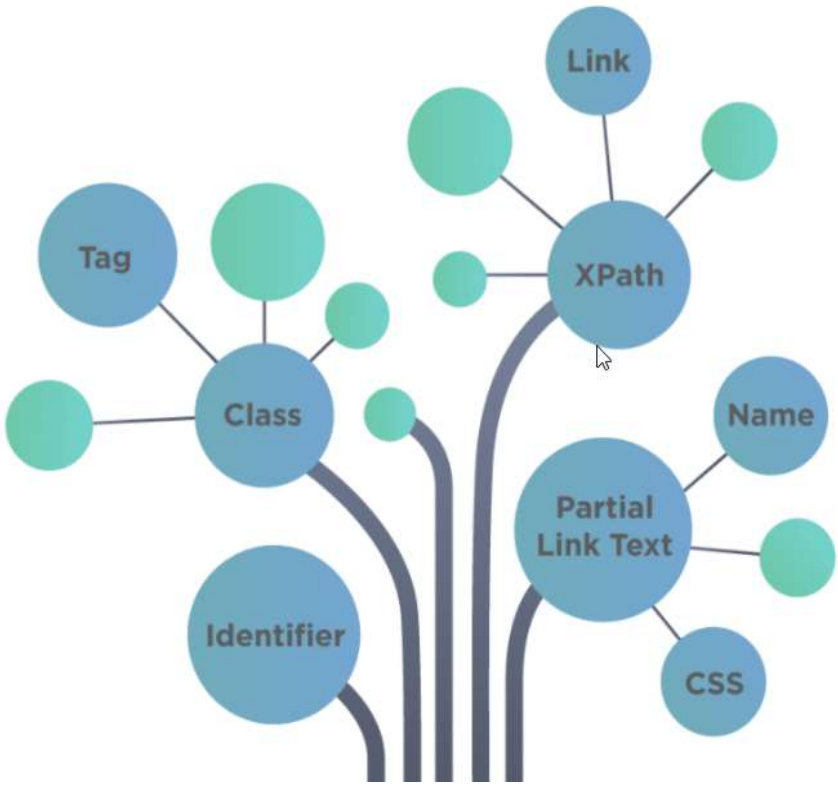


Рисунок 2.7 – Типи локаторів

ClassName використовує атрибут класу для ідентифікації об’єкта.

cssSelector використовується для створення правил стилю для веб-сторінок і може використовуватися для ідентифікації будь-якого веб-елемента.

Id може ідентифікувати елементи за допомогою атрибута 'id'.

linkText – це текст, який використовується в гіперпосиланнях, також може знаходити елемент.

name використовує атрибут Name для ідентифікації елементу.

partialLinkText – це частина тексту в посиланні яка також може ідентифікувати елемент.

tagName використовує ім’я теуг для пошуку елементів.

XPath (XML Path Language) — мова запитів до елементів XML або XHTML документа, що реалізує навігацію DOM. Призначений для використання іншими специфікаціями. XPath використовує вираз XML, щоб знайти елемент на веб-сторінці. Подібно до селекторів CSS, Xpath дуже корисний для пошуку

динамічних елементів на веб-сторінці. Може отримати доступ до будь-якого елемента на веб-сторінці, навіть якщо він має динамічні властивості.

Основний синтаксис ідентифікації веб-елемента за допомогою стратегії локатора XPath на рис. 2.8.

```
//tag_name[@attribute_value]
```

Рисунок 2.8 – Синтаксис ідентифікації веб-елемента за допомогою Xpath

На рис. 2.8 tag_name виступає у ролі імені тегу в структурі DOM, коли @attribute_value — це атрибут цільового елемента, який може однозначно ідентифікувати веб-елемент.

Тепер взаємодія з будь-яким веб-додатком вимагає, щоб драйвер Selenium ідентифікував веб-елементи на сторінці. Поки елемент не буде правильно ідентифікований, неможливо ініціювати будь-які дії над ним.

2.4 SpecFlow

Specflow — це платформа для тестування, яка підтримує практику BDD у платформі .NET. Це фреймворк з відкритим вихідним кодом, розміщений на GitHub. Він допомагає використовувати ATDD (розробка драйвера тесту прийняття) для додатків .NET. Завдяки цьому можна визначати тестовий сценарій англійською мовою Gherkin [19].

BDD (Behavior Driven Development) – це набір практик або підхід, подібний до TDD (Test Driven Development), який спрямований на подолання комунікаційного розриву між різними зацікавленими сторонами, такими як продукт, розробники та тестувальники.

Кінцева мета підходу BDD полягає в тому, щоб створити бізнес-вимоги, які могли б бути зрозумілі всій команді, щоб уникнути непорозумінь.

Feature Files – оскільки основна передумова BDD полягає в створенні легко зрозумілих тестів без зосередження на технічних аспектах, файли функцій написані мовою Gherkin. Один файл функції складається з функції та 'N' кількості сценаріїв, тобто одна функція розбивається на кілька сценаріїв.

SpecFlow (BDD Interpreter) – після того, як файл функцій (*.feature) буде готовий, його вміст має бути проаналізовано інтерпретатором BDD, який може розуміти формат файлів функцій.

На рис. 2.9 наведена взаємодія SpecFlow та BDD з веб-додатком [19].

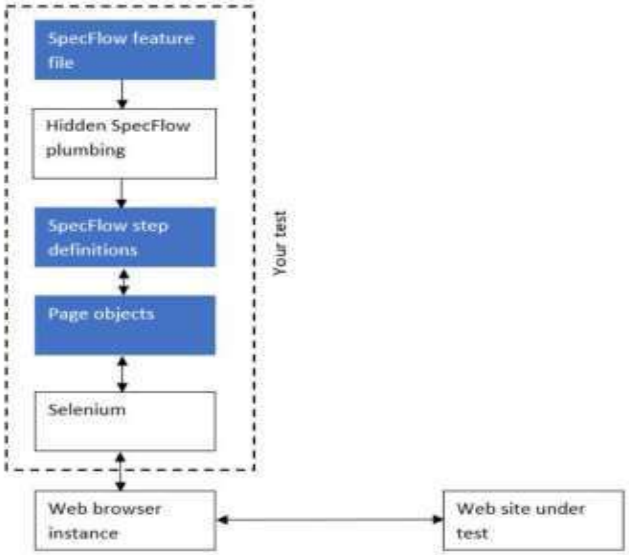


Рисунок 2.9 – Взаємодія SpecFlow та BDD з веб-додатком

Тести автоматизації (фреймворк NUnit + Selenium WebDriver) – містять реалізацію тесту, що відповідає кожному сценарію. Кожен крок сценарію матиме відповідне визначення кроку у файлі, де реалізовані тести автоматизації. У свою чергу, кожне визначення кроку матиме відповідний метод/реалізацію коду, до якого воно прив’язане.

Page Object Model - це шаблон проектування, який широко використовується спільнотою Selenium для автоматизованих тестів. Основний принцип проектування, якому дотримується об’єктна модель сторінки в

Selenium, полягає в тому, що центральне сховище об’єктів має бути створено для елементів керування на веб-сторінці. Отже, кожна веб-сторінка буде представлена окремим класом.

Об’єкти сторінки (або класи сторінки) містять елементи відповідної веб-сторінки разом із необхідними методами для доступу до елементів на сторінці. Отже, реалізація автоматизації тестування Selenium, яка використовує об’єктну модель сторінки в Selenium, буде складати різні класи для кожної веб-сторінки, що спрощує обслуговування коду.

Наприклад, якщо потрібно виконати автоматизацію сторінки входу та сторінки виписки, реалізація матиме клас для входу та виписки. Елементи керування для сторінки входу знаходяться в класі "сторінка входу", а елементи керування для сторінки виписки - в класі "сторінка виписки".

2.5 Засоби модульного тестування веб-додатків

NUnit – відкрите середовище модульного тестування застосунків для .NET. Воно було перенесене з мови Java (бібліотека JUnit) [20].

JUnit - це фреймворк з відкритим кодом, який використовується для написання та виконання модульних тестів мовою програмування Java. Це одна з найвідоміших платформ модульного тестування. Перевага даного підходу в ізолюванні окремо взятого модуля від інших.

Альтернативним фреймворком є TestNG, він призначений для тестування і поєднує в собі NUnit та JUnit.

Можливості:

- залежні методи для тестування серверних додатків
- підтримується в Eclipse, IDEA, Ant, Maven, Netbean

3 РОЗРОБКА ФРЕЙМВОРКУ АВТОМАТИЗАЦІЇ UI ВЕБ-ДОДАТКУ

3.1 Структура та компоненти

Компоненти необхідні для розробки фреймворку наступні:

1. .NET Core 3.1 - безкоштовна керована платформа програмного забезпечення з відкритим вихідним кодом.

2. SpecFlow - платформа BDD для .NET.

3. NUnit - платформа модульного тестування для всієї мови .Net.

4. Selenium WebDriver - інструмент для керування браузерами.

Структура фреймворку наступна:

1. appsettings.json – відповідає за налаштування локального запуску.

2. CommonPageMap.cs - клас із методами взаємодії із загальними елементами на WEB-сторінці. Методи доступні для всіх сторінок.

3. CommonPageAssertions.cs - загальні методи порівняння очікуваного та актуального результату. Методи доступні для всіх сторінок.

4. Configuration - містить класи з моделями даних для appsettings.json.

5. DriverManager.cs - клас, який надає екземпляр WebDriver для тестового запуску. Містить налаштування кількості паралельних потоків.

6. DriverFactory.cs - ініціалізує необхідну версію WebDriver. Підтримує Chrome, Firefox, Edge.

7. OptionsFactory.cs - постачальник необхідних параметрів для WebDriver.

8. DataManager.cs - надає методи для налаштування та отримання даних через ObjectContainer для кожного потоку.

9. WaitHelper.cs - методи очікування WebElements або деяких подій.

10. ConfigurationLoader.cs - завантажує налаштування з appsettings.json.

11. POM - папка для класів об'єктної моделі сторінок.

12. Features - папка для файлів функцій зі сценаріями тестування.

13. StepDefinitions - папка для класів з методами, які реалізують кроки з файлів функцій.

Для прикладу створимо тест для перевірки реєстрації у веб-додатку.

1. У .feature файлі створемо сценарій тестування реєстрації у веб-додаток інтернет-магазину. На рис. 3.1 наведено приклад тестового сценарії реєстрації у веб-додаток.

SignUp.feature

```
Feature: Sign Up
As a user
I want to be able to sign in
So that I have an ability to do shopping

Background:
    Given I navigate to the main page on the site

@Sign in
Scenario: As a user I want to be able to create an account
    When I navigate to the authentication page
    And I create a user with the following information
        | Field               | Value                |
        | Email               | mikhailmoroz2@gmail.com |
        | Personal First Name | Mikhail              |
        | Personal Last Name  | Moroz                |
        | Password            | Qqwerty1!!           |
        | Day of Birth        | 18                   |
        | Month of Birth      | May                  |
        | Year of Birth       | 1999                 |
        | Address First Name  | Mikhail              |
        | Address Last Name   | Moroz                |
        | Address             | St Main              |
        | City                | Kharkiv              |
        | State               | California           |
        | Zip code            | 62002                |
        | Phone               | 0991231232           |
    Then I should see 'Mikhail Moroz' user is registered
```

Рисунок 3.1 – Тестовий сценарій реєстрації у веб-додаток

Тестовий сценарій містить кроки навігації на головну сторінку у веб-додаток інтернет-магазин. Перехід на сторінку аутентифікації користувача. Крок створення користувача з тестовими даними та перевірку на те, що користувач успішно зареєстрований. Кроки які виділені у фіолетовий колір, ще не мають реалізації у класі визначення кроків. Далі потрібно згенерувати клас визначення кроків із шаблонами методів.

2. Створення класу визначення кроків із шаблонами методів наведені на рис. 3.2 та 3.3.

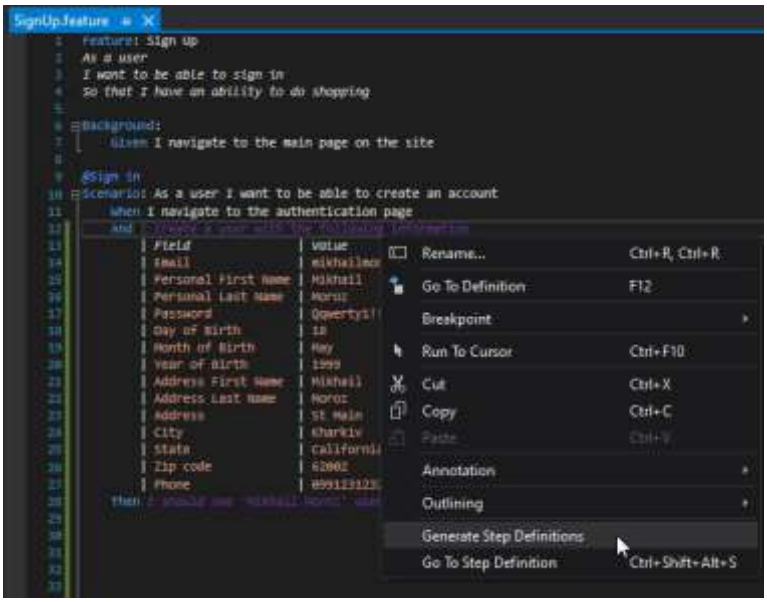


Рисунок 3.2 – Генерація класу визначених кроків з шаблонами методів

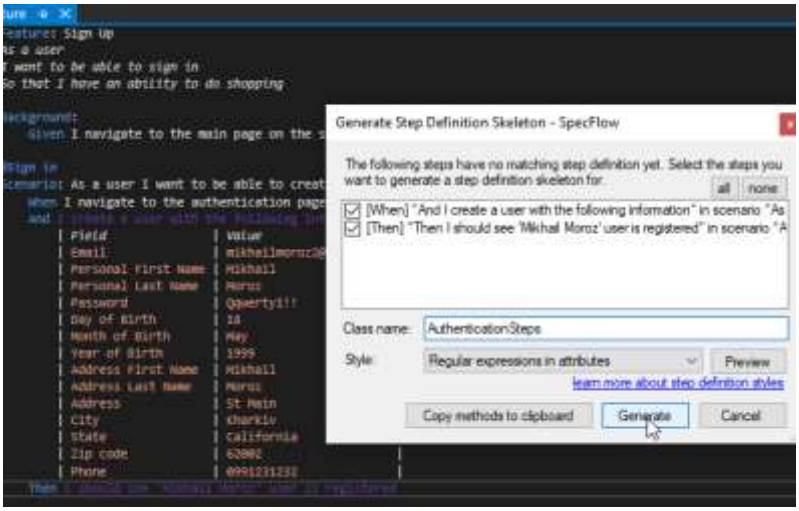


Рисунок 3.3 - Генерація класу визначених кроків з шаблонами методів

На рис. 3.4 можна побачити згенерований клас визначених кроків з шаблонами методів.

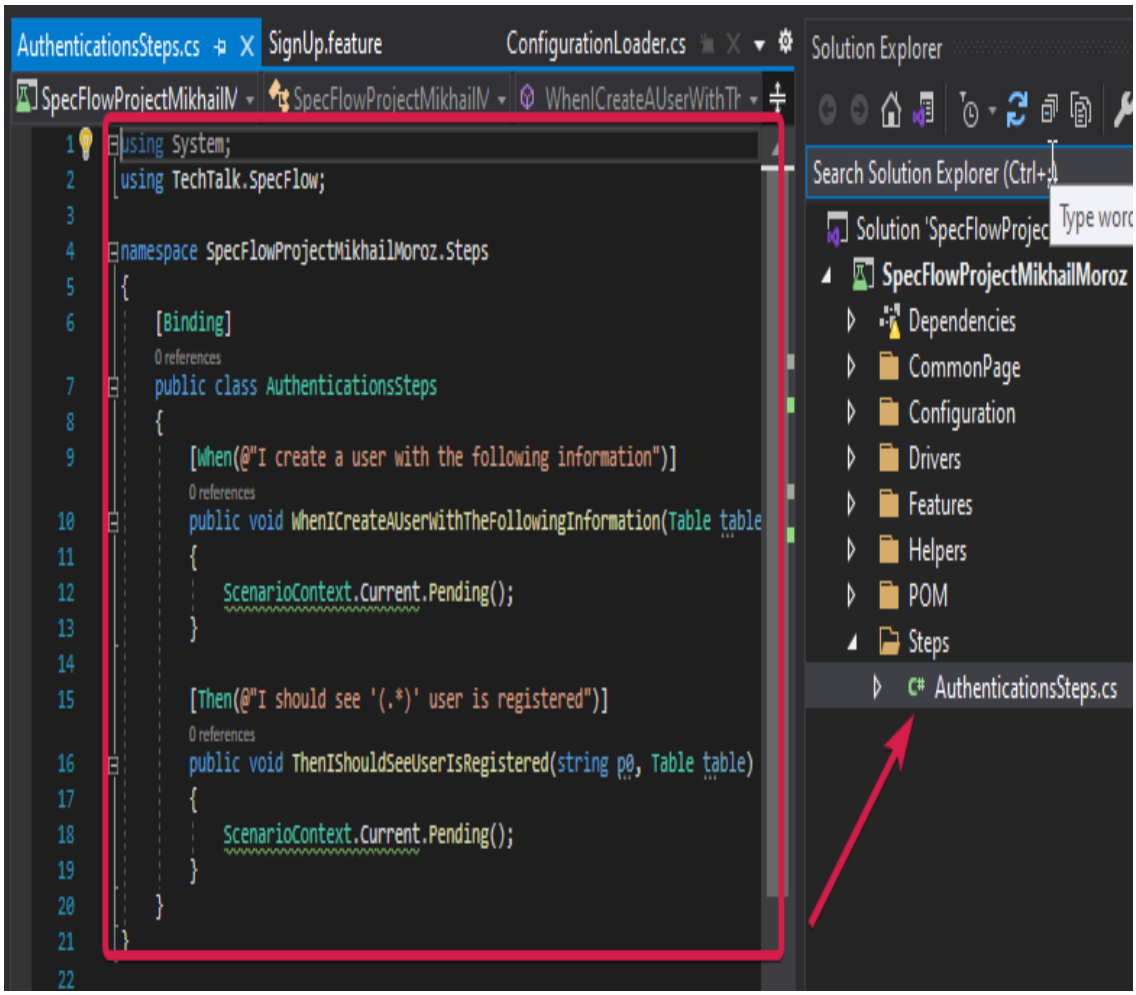
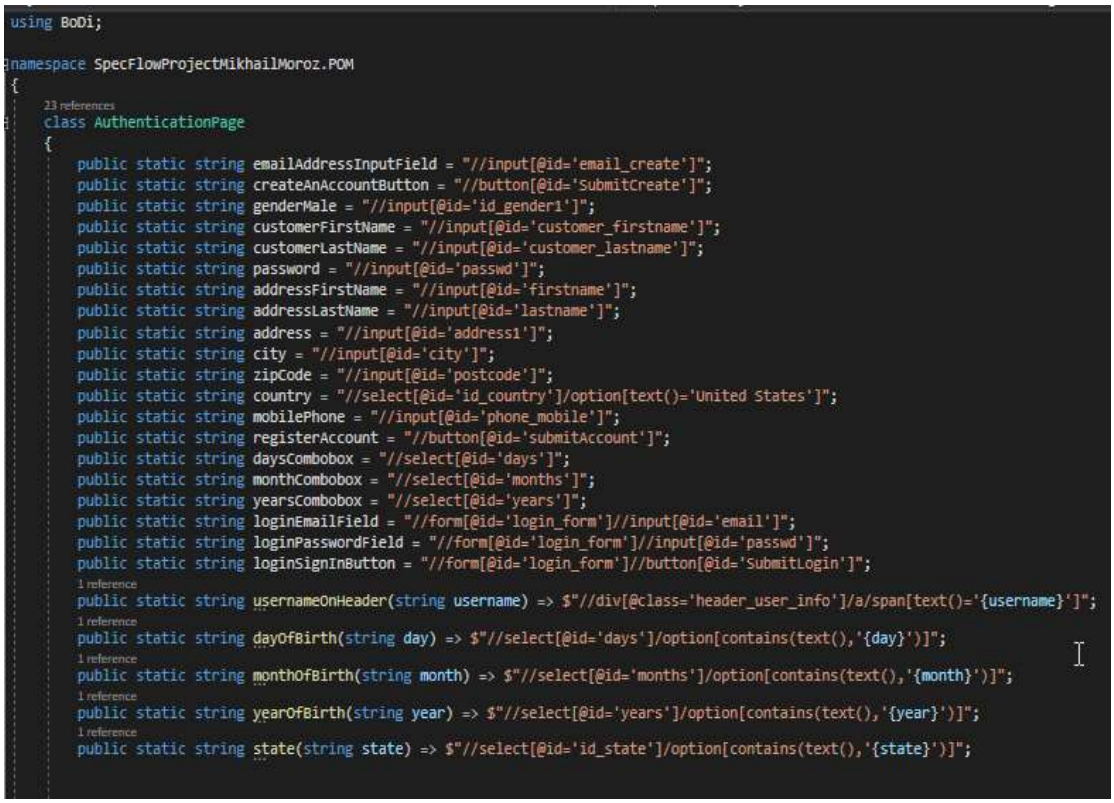


Рисунок 3.4 – Згенерований клас визначених кроків з шаблонами методів

Був згенерований клас визначення кроків із шаблонами методів у папці Steps з назвою AuthenticationSteps.cs. Де видно назву згенерованих кроків з шалонами методів, які були раніше виділені у фіолетовий колір.

3. Створення класу POM для реєстрації користувача у веб-додаток та додавання до нього необхідні локатори елементів, які наведені на рис. 3.5.



```
using BoDi;

namespace SpecFlowProjectMikhailMoroz.POM
{
    23 references
    class AuthenticationPage
    {
        public static string emailAddressInputField = "//input[@id='email_create']";
        public static string createAnAccountButton = "//button[@id='SubmitCreate']";
        public static string genderMale = "//input[@id='id_gender1']";
        public static string customerFirstName = "//input[@id='customer_firstname']";
        public static string customerLastName = "//input[@id='customer_lastname']";
        public static string password = "//input[@id='passwd']";
        public static string addressFirstName = "//input[@id='firstname']";
        public static string addressLastName = "//input[@id='lastname']";
        public static string address = "//input[@id='address1']";
        public static string city = "//input[@id='city']";
        public static string zipCode = "//input[@id='postcode']";
        public static string country = "//select[@id='id_country']/option[text()='United States']";
        public static string mobilePhone = "//input[@id='phone_mobile']";
        public static string registerAccount = "//button[@id='submitAccount']";
        public static string daysCombobox = "//select[@id='days']";
        public static string monthCombobox = "//select[@id='months']";
        public static string yearsCombobox = "//select[@id='years']";
        public static string loginEmailField = "//form[@id='login_form']//input[@id='email']";
        public static string loginPasswordField = "//form[@id='login_form']//input[@id='passwd']";
        public static string loginSignInButton = "//form[@id='login_form']//button[@id='SubmitLogin']";
        1 reference
        public static string usernameOnHeader(string username) => $"//div[@class='header_user_info']/a/span[text()='{username}']";
        1 reference
        public static string dayOfBirth(string day) => $"//select[@id='days']/option[contains(text(),'{day}')]";
        1 reference
        public static string monthOfBirth(string month) => $"//select[@id='months']/option[contains(text(),'{month}')]";
        1 reference
        public static string yearOfBirth(string year) => $"//select[@id='years']/option[contains(text(),'{year}')]";
        1 reference
        public static string state(string state) => $"//select[@id='id_state']/option[contains(text(),'{state}')]";
```

Рисунок 3.5 – Локатори необхідних елементів для реєстрації у веб-додаток

На рис. 3.5 зображені локатори необхідних елементів HTML з DOM-дерева, які будуть використовуватися для передачі їх у методи, які наведені на рис. 3.6.

```
0 references
public AuthenticationPage SendTextToEmailField(string emailAddress)
{
    _authenticationMap.SendTextToInput(emailAddressInputField, emailAddress);
    return this;
}
0 references
public AuthenticationPage ClickOnCreateAnAccount()
{
    _authenticationMap.ClickButton(createAnAccountButton);
    return this;
}
0 references
public AuthenticationPage SelectGender()
{
    _authenticationMap.ClickButton(genderMale);
    return this;
}
0 references
public AuthenticationPage SendTextToPersonalFirstNameField(string firstName)
{
    _authenticationMap.SendTextToInput(customerFirstName, firstName);
    return this;
}
0 references
public AuthenticationPage SendTextToPersonalLastNameField(string lastName)
{
    _authenticationMap.SendTextToInput(customerLastName, lastName);
    return this;
}
0 references
public AuthenticationPage SendTextToPasswordField(string userPassword)
{
    _authenticationMap.SendTextToInput(password, userPassword);
    return this;
}

0 references
public AuthenticationPage SelectDateOfBirth(string day, string month, string year)
{
    _authenticationMap.ClickButton(dayOfBirth(day));
    _authenticationMap.ClickButton(monthOfBirth(month));
    _authenticationMap.ClickButton(yearOfBirth(year));
    return this;
}

0 references
public AuthenticationPage SendTextToAddresslFirstNameField(string firstName)
{
    _authenticationMap.SendTextToInput(addressFirstName, firstName);
    return this;
}

0 references
public AuthenticationPage SendTextToAddresslLastNameField(string lastName)
{
    _authenticationMap.SendTextToInput(addressLastName, lastName);
    return this;
}

0 references
public AuthenticationPage SendTextToAddressField(string addressName)
{
    _authenticationMap.SendTextToInput(address, addressName);
    return this;
```

Рисунок 3.6 – Методи класу AuthenticationPage.cs

На рис. 3.6 зображені реалізовані методи класу AuthenticationPage, які будуть використовуватися для реалізації методів визначення кроків у класі AuthenticationSteps.cs наведені на рис. 3.7.

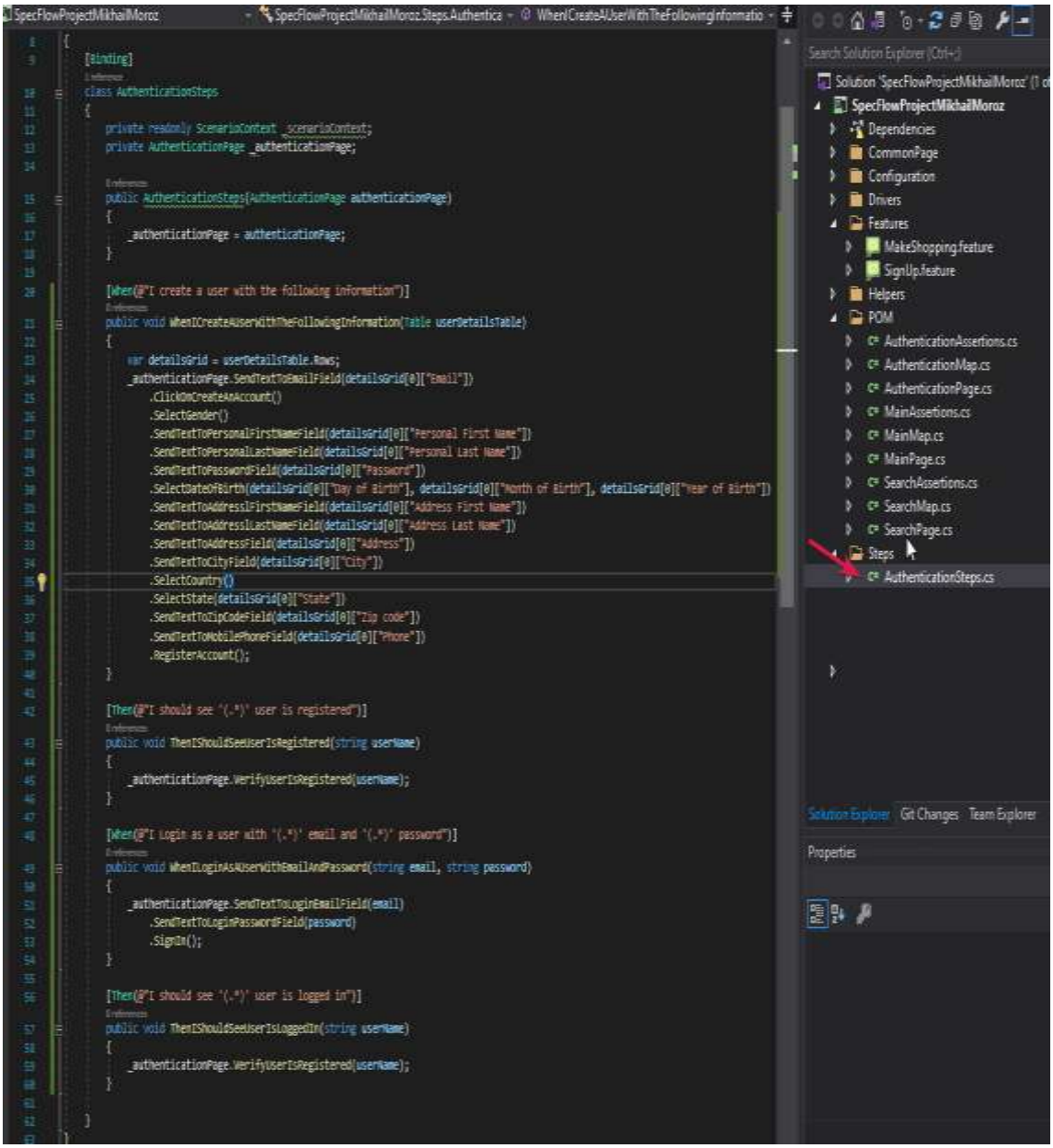


Рисунок 3.7 – Реалізація методів визначення кроків класу AuthenticationSteps

4. Далі компілюємо рішення, відкриваємо Test Explorer та запускаємо тестовий сценарій. На рис. 3.8 зображений Test Explorer з компільованим рішенням та успішно пройденим тестовим сценарієм з реєстрації користувача у веб-додатку інтернет-магазину.

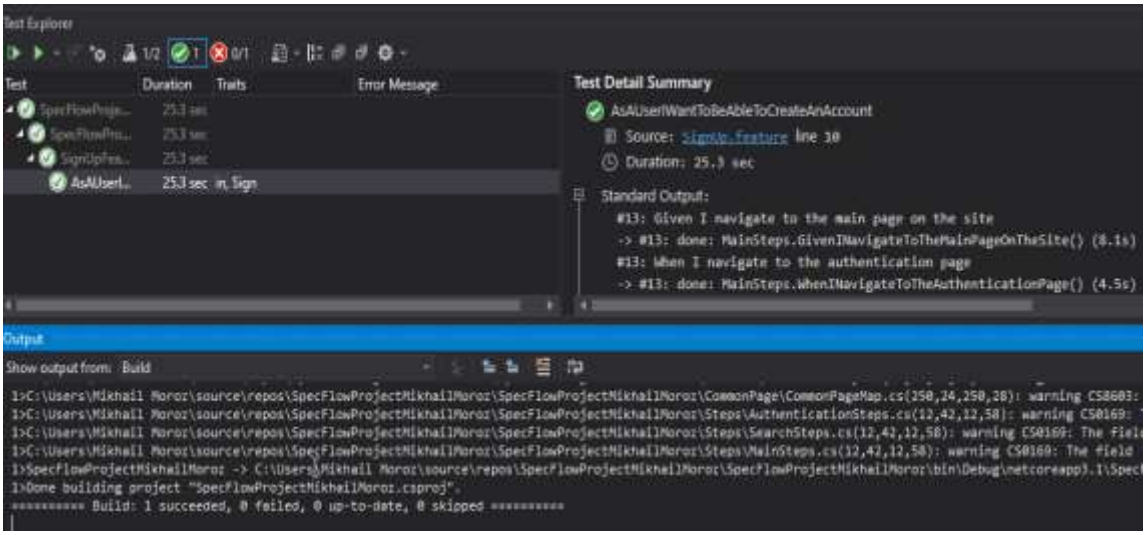


Рисунок 3.8 – Успішно скомпільований та пройдений тестовий сценарій

З результату можна побачити, що увесь процес реєстрації зайняв 25 секунд, коли було потрібно ввести велику кількість даних. У наступному підрозділі розглянуто більш детально саму структуру фреймворку.

3.2 Діаграма фреймворку та реалізація класів

Структура фреймворку яка була розглянута раніше, розглянемо на діаграмі, яку зображено на рис. 3.9.

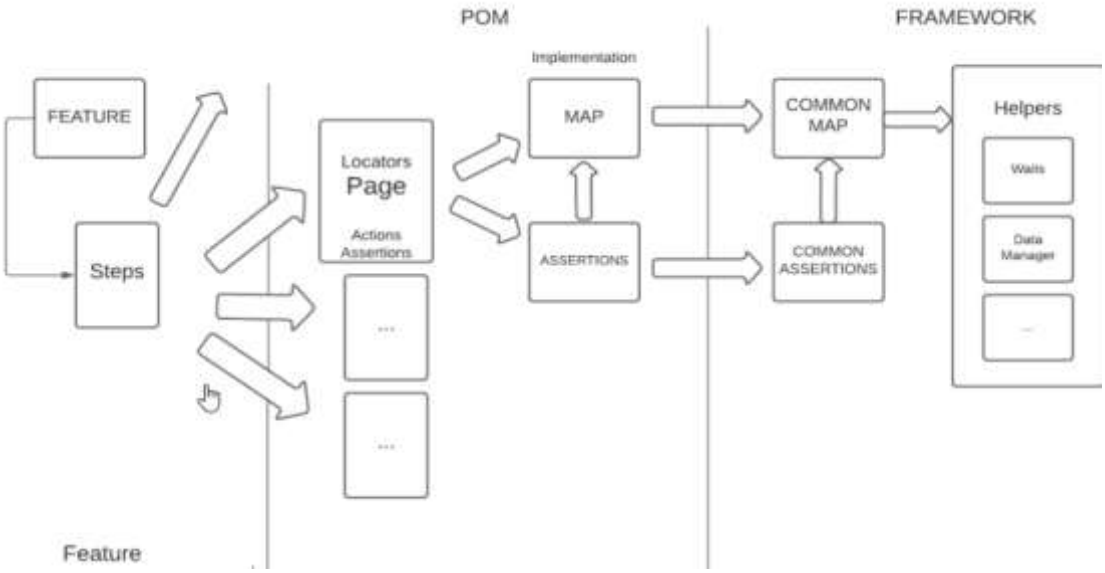


Рисунок 3.9 – Діаграма структури та взаємодії компонентів

Далі будуть наведені класи фреймворку, та їх призначення. На рис. 3.10 зображений клас DataManager.cs, який надає методи для налаштування та отримання даних через ObjectContainer для кожного потоку.

```
using BoDi;
using System;
using System.Collections.Generic;
using System.Text;

namespace SpecFlowProjectMikhailMoroz.Helpers
{
    1 reference
    class DataManager
    {
        private IObjectContainer _objectContainer;
        0 references
        public DataManager(IObjectContainer objectContainer)
        {
            _objectContainer = objectContainer;
        }

        0 references
        public void SetData(object data, string name)
        {
            _objectContainer.RegisterInstanceAs<object>(data, name);
        }

        0 references
        public object GetData(string name)
        {
            return _objectContainer.Resolve<object>(name);
        }

        0 references
        public void SetList(List<string> data, string name)
        {
            _objectContainer.RegisterInstanceAs<List<string>>(data, name);
        }

        0 references
        public List<string> GetList(string name)
        {
            return _objectContainer.Resolve<List<string>>(name);
        }

        0 references
        public void SetListOfDict(List<Dictionary<string, string>> data, string name)
        {
            _objectContainer.RegisterInstanceAs<List<Dictionary<string, string>>>(data, name);
        }

        0 references
        public List<Dictionary<string, string>> GetListOfDict(string name)
        {
            return _objectContainer.Resolve<List<Dictionary<string, string>>>(name);
        }

        0 references
        public void SetDict(Dictionary<string, string> data, string name)
        {
            _objectContainer.RegisterInstanceAs<Dictionary<string, string>>(data, name);
        }

        0 references
        public Dictionary<string, string> GetDict(string name)
        {
            return _objectContainer.Resolve<Dictionary<string, string>>(name);
        }
        0 references
```

Рисунок 3.10 – Реалізація класу DataManager.cs

На рис. 3.11 зображений клас WaitHelper.cs, який містить методи очікування WebElements.

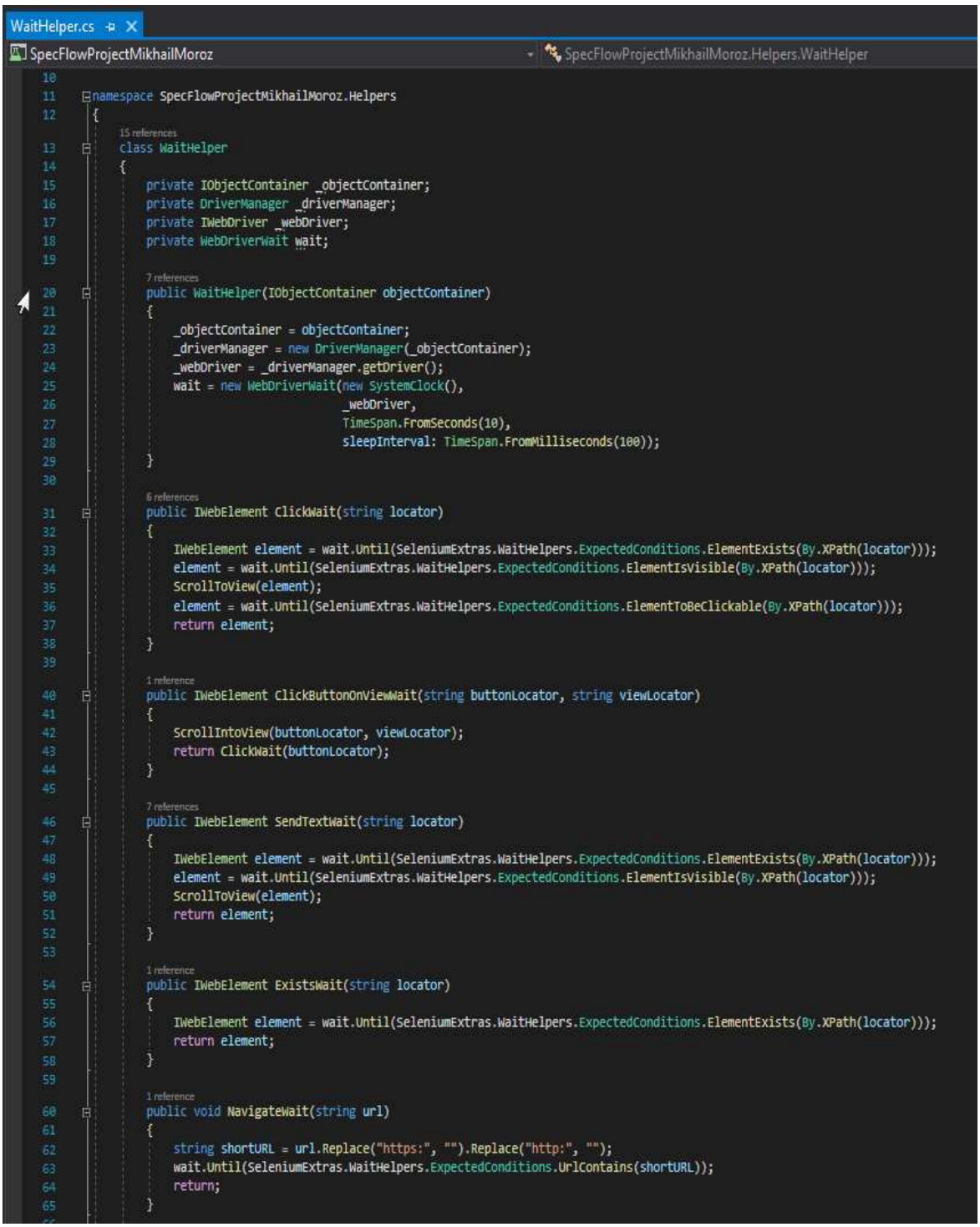


Рисунок 3.11 – Реалізація класу WaitHelper.cs

На рис. 3.12 та 3.13 наведений клас CommonPageMap.cs з методами взаємодії із загальними елементами на WEB-сторінці. Методи доступні для всіх сторінок.

Рисунок 3.12 – Реалізація класу CommonPageMap.cs

Рисунок 3.13 - Реалізація класу CommonPageMap.cs

На рис. 3.14 зображений клас CommonPageAssertions.cs, який містить загальні методи порівняння очікуваного та актуального результату.

Рисунок 3.14 – Реалізація класу CommonPageAssertions.cs

На рис. 3.15 зображена реалізація класу DriverManager.cs, який надає екземпляр WebDriver для тестового запуску.

Рисунок 3.15 – Реалізація класу DriverManager.cs

На рис. 3.16 зображена реалізація класу DriverFactory.cs, який ініціалізує необхідну версію WebDriver.

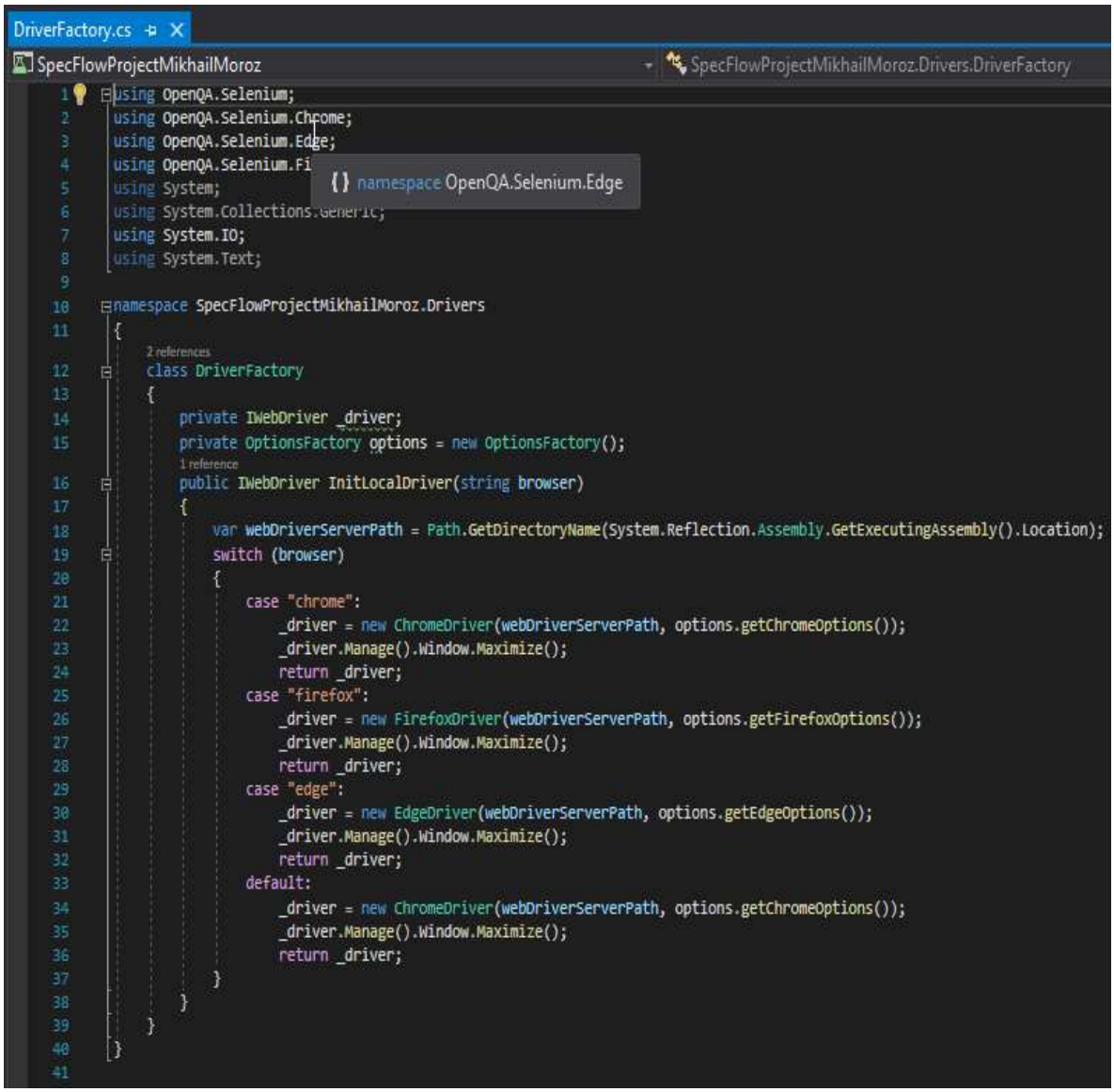


Рисунок 3.16 – Реалізація класу DriverFactory.cs

На рис. 3.17 зображена реалізація класу OptionsFactory.cs, який постачає необхідні параметри для WebDriver.

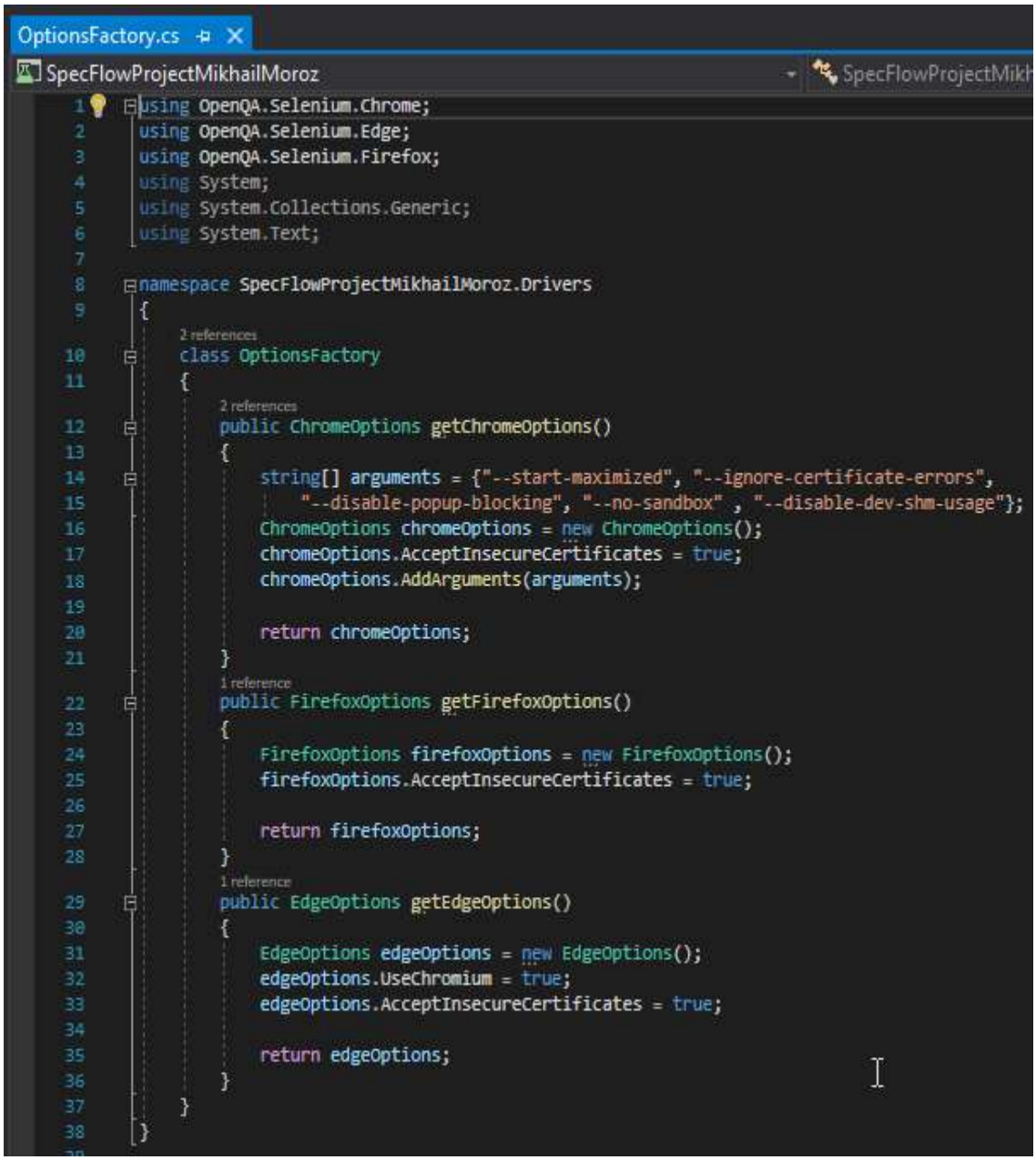


Рисунок 3.17 – Реалізація класу OptionsDactory.cs

Для реалізації фреймворку були розроблені основні класи з методами взаємодії елементів у веб-браузері, допоміжні класи з методами очікування веб-елементів, методи порівняння очікуваного та актуального результату, методи налаштування та отримання даних.

ВИСНОВКИ

Метою магістерської кваліфікаційної роботи було дослідження та розробка фреймворку для автоматизації взаємодії користувача з користувацьким інтерфейсом веб-додатку.

В ході роботи була проаналізована предметна область, яка пов’язана з забезпеченням якості та були досліджені існуючі засоби для автоматизованого тестування користувацького інтерфейсу веб-додатків.

Та результатом роботи є розроблений фреймворк для автоматизації тестування користувацького інтерфейсу веб-додатків, для якого були підібрані необхідні компоненти, розроблені та реалізовані різноманітні методи у класах для імітації дій користувачів у веб-додатку, налаштований запуск тестових сценаріїв в таких браузерах як Google Chrome, Edge, Firefox. Розроблено класи, які налаштовують необхідні параметри та конфігурації для WebDriver. Класи помічники такі як DataManager, який надає методи для налаштування та отримання даних через ObjectContainer для кожного потоку та клас помічник WaitHelper, який містить методи очікування веб-елементів. Тому, опираючись на постановку задачі, можна сказати що мета роботи були досягнута.

Розроблений фреймворк можно розвивати далі, наприклад, додати до нього клас, який буде створювати та надсилати звіти, наприклад до TestRail. Додати клас, який буде генерувати локальні звіти про тестовий запуск, створити, клас який налаштовуватиме запуск тестових сценаріїв для віддаленого доступу на різноманітних хостах з різними конфігураціями і так далі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Жизненный цикл ПО // [Електронний ресурс] – Режим доступу: https://qalight.ua/ru/baza-znaniy/zhiznennyiy-tsikl-po/

2. Международные стандарты качества // [Електронний ресурс] – Режим доступу: https://friedman.com.ua/info/tariff-regulation/certifications/mezhdunarodnye-standarty-po-upravleniju-iso-9000-2000-v-ukraine-343/

3. Качество ПО // [Електронний ресурс] – Режим доступу: https://qalight.ua/ru/baza-znaniy/kachestvo-programmnogo-obespecheniya/

4. Приемочное тестирование // [Електронний ресурс] – Режим доступу: https://qalight.ua/ru/baza-znaniy/priemochnoe-testirovanie/

5. Системное тестирование // [Електронний ресурс] – Режим доступу: https://qalight.ua/ru/baza-znaniy/sistemnoe-testirovanie/

6. Интеграционное тестирование // [Електронний ресурс] – Режим доступу: https://qalight.ua/ru/baza-znaniy/integratsionnoe-testirovanie/

7. Модульное тестирование // [Електронний ресурс] – Режим доступу: https://qalight.ua/ru/baza-znaniy/modulnoe-testirovanie/

8. Методология Agile // [Електронний ресурс] – Режим доступу: https://scrumtrek.ru/blog/agile-scrum/4029/metodologiya-agile/

9. Модель Waterfall // [Електронний ресурс] – Режим доступу: https://changellenge.com/article/ne-tolko-agile-kak-ustroena-model-waterfall-i-v-kakikh-proektakh-ee-ispolzovat/

10. Обеспечение качества // [Електронний ресурс] – Режим доступу: http://www.protesting.ru/qa/

11. Фундаментальний процес тестирования // [Електронний ресурс] – Режим доступу: https://qalight.ua/baza-znaniy/fundamentalnij-protses-testuvannya/

12. Особливості веб-додатків // [Електронний ресурс] – Режим доступу: http://sites.znu.edu.ua/webprog/lect/1191.ukr.html

13. Мінухін С.В. Дослідження засобів створення обчислювального кластера на основі технологій віртуалізації // Інформаційно-керуючі системи на залізничному транспорті. – 2016. – № 3. – С. 38–49.

14. Минухин С.В. Информационные технологии обработки заданий в распределенных вычислительных средах // Системи обробки інформації. — 2017. — № 2(148). – С.53–56.

15. Selenium // [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Selenium

16. Selenium WebDriver // [Електронний ресурс] – Режим доступу: https://www.browserstack.com/guide/selenium-webdriver-tutorial

17. Selenium locators // [Електронний ресурс] – Режим доступу: https://uk.myservername.com/selenium-locators-identify-web-elements-using- xpath-selenium

18. Положення локаторів у Chrome // [Електронний ресурс] – Режим доступу: https://uk.myservername.com/how-locate-elements-chrome

19. SpecFlow tutorial for automation testing // [Електронний ресурс] – Режим доступу: https://www.lambdatest.com/blog/specflow-tutorial-for-automation-testing/

20. NUnit // [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/NUnit