Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Система для автоматизації тестування
з використанням технології Selenium
(тема)

Виконав:
здобувач 4 року навчання,
групи КІУКІ-21-6
Карина ГОЛОВАН
(власне ім’я, прізвище)

Спеціальність
123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

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

Освітня програма
Комп’ютерна інженерія
(повна назва освітньої програми)

Керівник: ас. Олег ЖУРИЛО
(посада, власне ім’я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім’я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва)

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

Освітня програма Комп’ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Голован Карині Василівні
(прізвище, ім’я, по батькові)

1. Тема роботи Система для автоматизації тестування з використанням технології Selenium

затверджена наказом по університету від “ 26 ” травня 2025 р. № 424Ст

2. Термін подання студентом роботи до екзаменаційної комісії 17 червня 2025р

3. Вхідні дані до роботи

1. Фреймворк автоматизованого тестування Selenium
2. Середовище для запису тестів Selenium IDE
3. Мова програмування Java
4. Документація та довідкові матеріали щодо Selenium

4. Перелік питань, що потрібно опрацювати у роботі

1. Огляд та аналіз джерел інформації з теми автоматизованого тестування вебзастосунків
2. Аналіз та огляд існуючих рішень та аналогів
3. Дослідження фреймворку Selenium та його основних функцій
4. Огляд Selenium IDE та характеристика вебзастосунку, що тестується
5. Проєктування та реалізація автоматизованих тестів для перевірки основного
6. Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Слайд презентація - 10 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

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

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	27.05.2025	
2	Проведення аналізу існуючих матеріалів	27.05- 04.06.2025	
3	Формування плану роботи та визначення основних етапів	04.06.2025	
4	Реалізація поставлених задач	05.06.2025 – 09.06.2025	
5	Аналіз отриманих результатів та формування рекомендацій для покращення	09.06.2025-10.06.2025	
6	Оформлення пояснювальної записки	10.06.2025 – 11.06.2025	

Дата видачі завдання 26 травня 2025 р.

Студент _____ (підпис)

Керівник роботи _____ (підпис) ас. Олег ЖУРИЛО (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 73 с., 27 рис., 6 табл., 1 дод., 7 джерел.

SELENIUM, SELENIUM IDE, SELENIUM WEBDRIVER, JAVA, JUNIT5, ВЕБ-ЗАСТОСУНОК, ТЕСТУВАННЯ, ТЕСТОВИЙ СЦЕНАРІЙ.

Метою кваліфікаційної роботи є вивчення, дослідження та аналіз основних методів та принципів у автоматизованому тестуванні для веб-застосунків із використанням інструментів Selenium.

У ході виконання кваліфікаційної роботи було вивчено та проаналізовано новітні підходи у автоматизації веб-застосунків, а також, було вивчено можливості фреймворку Selenium, а також необхідні інструменти, а саме: Selenium IDE, Selenium WebDiver, Junit 5. Основний фокус був на проведенні тестування, створенні стратегії тестування, сценаріїв, створенню автоматизованих тестів для веб-застосунку з розрахунку вартості розмитнення. У результаті це дало можливість оцінити стабільність роботи при різних сценаріях використання.

ABSTRACT

Bachelor’s thesis: 73 pages, 27 figures, 6 tables, 1 appendices, 7 sources.

SELENIUM, SELENIUM IDE, SELENIUM WEBDRIVER, JAVA, JUNIT5, WEB APPLICATION, TESTING, TEST SCRIPT.

The major goal of this thesis is of the qualification project is to explore, research, and analyze the fundamental methods and principles of automated testing for web applications using Selenium tools.

In order to achieve this goal, modern approaches to web application test automation were studied and analyzed. Additionally, the capabilities of the Selenium framework and essential tools such as Selenium IDE, Selenium WebDriver, and JUnit 5 were examined. The main focus was on executing tests, designing a testing strategy, developing scenarios, and creating automated test cases for a web application that calculates customs clearance costs. As a result, the testing process made it possible to evaluate the application's stability and reliability under different usage scenarios.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

Selenium – це фреймфорк, який розроблений спеціально для автоматизації тестування веб-застосунків, він дозволяє імітувати взаємодію користувача з браузером

Тестовий сценарій – це опис покроковості дій в тестуванні за допомогою яких виконується перевірка конкретного функціонування продукту

Тестовий набір – об’єднана колекція тестових сценаріїв , які запускаються разом для перевірки повної системи або ж певної частини системи

Запис тестових сценаріїв – це процес створення автоматизованих тестів за якими будуть проводитися автоматизовані тести, а саме, шляхом фіксації дій користувача у браузері: кліки, введення даних, навігацію та перевірки

Selenium IDE – це спеціальне розширення для браузера, за допомогою нього отримується дозвіл записувати, редагувати та запускати автоматизовані тести без написання коду

Selenium WebDriver – інструмент створений спеціально для управління браузером на програмному рівні, він дозволяє створити складні тести за допомогою мов програмування

JUnit 5 – це сучасний фреймворк який допомагає у модульному тестуванні на Java, він надає можливість створювати та виконувати та перевіряти тести

Калькулятор розмитнення – спеціальний сервіс, за допомогою якого користувач може розрахувати вартість митних платежів для певного транспортного засобу

ВСТУП

Сучасний світ сповнений новітніми та сучасними цифровими технологіями і всі вони дуже швидко проникають у всі сфери нашого життя, веб- застосунки стають все більш популярними та є основним інструментом у продажах, наданні послуг та загальній взаємодії з користувачами. Саме через це питання якості у програмному забезпеченні виходить на перший план та набуває ще більшого значення. Ключовими факторами є відсутність помилок, зручність використання та стабільність у роботі - всі ці критерії безпосередньо впливають на довіру користувачів.

Саме для цих цілей є одним з найдієвіших підходів у забезпеченні якості є тестування. Однак, звичне нам ручне тестування має недоліки, адже для його проведення витрачається багато часу, ресурсів, а також в ньому присутній людський фактор, а це може призвести до пропуску важливих помилок, повторюваності в результатах. На поміч нам приходить автоматизація тестування, яка стала надважливою у сучасному процесі розробки програмного забезпечення. За допомогою автоматизованих тестів можна значно скоротити час на виконання тестових сценаріїв, а також оптимізувати час та витрату людського ресурсу, що дозволить підвищити контроль якості та загальну ефективність і напряму впливатиме на кінцевий продукт та оцінку від кінцевих користувачів.

У цій кваліфікаційній роботі основне зосередження буде на дослідженні можливостей та практичному застосуванні автоматизованого тестування до веб-застосунку. У якості основного інструменту для дослідження мною було обрано потужний фреймворк Selenium, бо він себе показав як одне з найефективніших рішень для автоматизації взаємодії з веб – застосунками. Для практичної частини було обрано актуальний та важливий під час купівлі автомобілів калькулятор, який призначений для здійснення розрахунків митних платежів при ввезені авто або іншого типу

транспорту. Обрано саме цей калькулятор, бо він несе безпосередню цінність для людей які планують купівлю авто або ж планують використовувати його для розвитку власного бізнесу та здійснювати імпорт транспорту.

На результатах проведеного мною дослідження я планую показати, що Selenium є ефективним інструментом для тестування та забезпечення високої якості та надійності роботи веб-застосунків, а також з важливістю математичних та фінансових розрахунків, які потребують особливої точності.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд літературних джерел

Selenium — це один з сучасних інструментів який використовується для автоматизації тестування веб-застосунків, він використовується тестувальниками та безпосередньо напряму розробниками. Фреймворк надає нам максимально обширний вибір функцій, які предназначені для створення, запуску та аналізу тестів, саме це дозволяє виконувати автоматизацію для перевірки коректності роботи веб-інтерфейсів [1].

За домогою Selenium можна протестувати велику кількість різних аспектів у функціональності, до основних таких типів можна віднести:

Перевірку елементів інтерфейсу: можна перевірити доступність різних елементів, наприклад, як після взаємодії змінюються елементи, чи вірно відображаються наявні поля, блоки чи кнопки, а також динамічні елементи.

Перевірку валідації: є можливість перевірити чи працює валідація якщо ввести невірні данні у спеціальні поля, протестувати обов’язкові поля на введення, вірність форматування, наприклад якщо у веб-застосунку яеий тестується треба ввести номер телефону, чи емейл або ж навіть якусь дату у конкретному форматі.

Функціональне тестування: можливість перевірити чи вірно виконуються всі функції, наприклад чи коректні переходи між сторінками або ж як працює взаємодія між алертами, чи певними вікнами, також правильність роботи логіки та обчислень, що можна використати при тестуванні калькуляторів, чи якихось фільтрів, випадаючих списків, посилань або певних кнопок.

Тестування навігації: за допомогою цього виду тестування ми можемо перевірити чи вірно працює меню на сторінці, чи можливі переходи між сторінками, чи вірно переводить посилання. Це є дуже важливим у

користувацькому досвіді, бо зручність у користуванні є одним з найважливіших аспектів коректної роботи веб-сайту та дає можливість забезпечити вірну та цілісну структуру.

Динамічне тестування: наразі кожен веб-сайт наповнений різноманітним контентом, а отже його потрібно протестувати на коректність роботи і в цьому нам допомагає Selenium, де можна перевірити наявні графіки або ж таблички, і навіть елементи які не одразу з'являються на сторінці.

Отже, так як він має широкі можливості для проведення тестування, то ми можемо покрити більшість необхідних для тестування сценаріїв, які допоможуть в подальшому вдосконалити взаємодію користувачів з веб-інтерфейсами. А отже він стає універсальним інфтрументом, який дозволяє виконувати прості перевірки, а також більш складні для повноцінного тестування. Також, хочу розглянути більш детальніше які саме можливості надає Selenium та як їх використовують.

Selenium має широкий ряд можливостей та аспектів, які підходять для автоматизованого тестування і до основних відносяться:

Підтримка різних браузерів: Selenium дозволяє нам виконувати кросбраузерне тестування для веб- застосунків. Бо Selenium WebDriver надає нам окремі драйвери для найпопулярніших браузерів, якими користуємось. А саме для Google Chrome використовуємо ChromeDriver, також для Mozilla Firefox – GeckoDriver, для Microsoft Edge же використовуємо EdgeDriver. А для Apple Safari використовується SafariDriver, але він активується за допомогою налаштування налаштування системи.

Розгляну детальніше Selenium WebDriver, який є основною складовою Selenium, а це дозволяє автоматизовувати взаємодію з браузером та напряму виконувати тести на сторінках. А завдяки ж WebDriver можна здійснювати на високому рівні контроль та взаємодіяти з різними елементами, наприклад, кнопками, посиланнями, писками та іншим. А також він надає можливість виконувати кліки- натискати на посилання, чек- бокси, кнопки, виконувати

введення тексту у поля, отримати значення елементів та взаємодіяти з іншими вкладками, діалоговими вікнами, тощо.

Підтримка різних мов програмування: Selenium підтримує найпопулярніші мови програмування і це безпосередньо робить його універсальним інструментом, який підходить для автоматизації тестування, а спеціалісту дозволяє вибрати зручну та зрозумілу для нього мову програмування, напридклад, Java – яка є одною з найпопулярніших мов для роботи з Selenium. Вона популярна, бо має великий набір бібліотек для тестування, а також багато переваг. Python – також одна з популярних мов для написання тестів, бо вона є читабельною та має обширний вибір бібліотек для тестування. C# - його також підтримує Selenium, а це робить його гарним вибором для виконання тестування в середовищі .NET. JavaScript – підтримується через Node.js, саме це дозволяє спеціалістам написати тести для веб-застосунків. Selenium тепер можна використовувати разом із Mocha, Jasmine, або Cypress.

Маштабованість а також можливість виконувати паралельно необхідні тести: наразі для того аби максимально підвищити продуктивність та користь автоматизованого тестування, а також безпосередньо зменшити час який потребується на виконання одразу декількох тестів або й більше, то можливо Selenium маштабувати, зробити ж це можна за допомогою Selenium Grid. Це є спеціальне розширення яке використовується для Selenium WebDriver, воно дозволяє виконувати розподілений запуск тестів. Самі ж тести можна запускати одночасно на різних вузлах і це буде скорочувати загальний час, який потрібен для тестування, а також, можна виконувати кросплатформенне тестування і тестувати одразу на декількох різних браузерів або ж навіть на різних операційх системах. Завдяки цьому тестувальник або ж розробник зможе перевірити коректність роботи веб-сайту у різних середовищах, що є важливим для того аби перевірити та забезпечити сумісність.

Можливість перевірити умови та результати: Selenium має багато переваг, але ж однією з надважливих функцій є можливість виконувати

перевірки які визначать відповідність реального(фактичного) стану застосунку тому який був очікуваним. В цьому нас допомагає механізм який дозволить контролювати вірність значень, відображення обраних елементів, перевірки на наявність тексту та іншого. Загалом це буде значно знижувати ризики які є при пошуку помилок у середовищі, а також робить тестування більш ефективним.

Відкрить і доступність: у Selenium є можливість скористатись відкритим вихідним кодом, а це значно спрощує написання тестів, адже дозволяє тестувальникам і самим розробникам використовувати цей код удосконалюючи його, модифікуючи під свої потреби, ну і звичайно тим самим підвищувати його функціональність. Також, Selenium можна користуватися абсолютно безкоштовно та він не потребує спеціальної ліцензії на використання, саме завдяки цьому він набув широкої популярності та може використовуватися широкою кількістю людей для навчання та можна його впроваджувати для навчання в освітні заклади і тд.

Гнучкість при створенні тестів: Selenium не має жодних обмежень при створенні структури тестів, тобто людина яка створює тести зможе сама обрати як саме організувати сценарії, як релізовувати логування, чи наприклад обробку та використання коду, а також він може інтегруватися зі створеними архітектурними шаблонами.

Headless-режим: це такий режим, за допомогою якого Selenium може запускати браузер без візуального відображення графіки інтерфейсу. Тобто, всі дії які проводяться під час тестування вони будуть виконуватися у фоновому режимі, а це дозволяє зекономити ресурси та зробити тестування ще більш ефективним. Цей режим може бути корисним при виконанні масового паралельного тестування, бо тут треба досягати максимальної швидкості або ж наприклад може використовуватися у хмарних системах, бо тут треба економити ресурси і як раз цей рижим з цим відмінно справляється.

Можна перераховувати ще безліч переваг Selenium, наприклад, такі як можливість записувати сценарії, інтеграція з системою DevOps, підтримка

динамічного контенту, очікування, але, хотіла б перейти до основного висновку про Selenium – що це є один з найкращих та найпотужніших інструментів, який підходить для автоматизації. Через те, що він є гнучким, має можливість працювати з різноманітними доступними мовами програмування, а також використовувати для тестування різні браузери, то його можна вважати універсальним інструментом та рішенням який підійде для багатьох цілей [1].

Але, хочу сказати, що не зважаючи на всі його переваги, наприклад як можливість використовувати готовий відкритий код для власних цілей, він все ж таки є складним інструментом і для того аби вдало використовувати Selenium треба мати розуміння та загальні принципи мов програмування(обрати мову якою будуть написані тести), вміти працювати з новітніми технологіями і тоді можна повноцінно скористатися всіма можливостями, які надає Selenium та за допомогою цього забезпечити максимально можливу високу якість у автоматизованому тестуванні та допомогти у пришвидшенні основних процесів у розробці, тестуванні та налагодженні роботи. А так як він ще може виявляти основні помилки на початкових етапах це безпосередньо впливає на вартість розробки та продукту і допомагає здешевити його, ну і звичайно ж в кінцевому етапі впливає на зручність у використанні та надійність кінцевого продукту.

1.2 Постановка завдання та вимог

Метою моєї кваліфікаційної роботи є створення тестів за допомогою Selenium які допоможуть в автоматизації тестування калькулятору розмитнення автомобілів та інших видів техніки на AUTO.RIA. Це є складний вид калькулятору і є інтерактивним інструментом, за допомогою якого будь-який користувач зможе ввести параметри свого транспорту у спеціальну форму калькулятору та розрахувати орієнтовану вартість у розмитненні транспорту. Основним, що дозволить мені перевірити різні

сценарії та оцінити ефективність роботи калькулятору буде:

а) перевірка на вірність введення даних;

б) тестування доступних форматів обчислень;

в) перевірка на коректність взаємодії з основними елементами інтерфейсу;

г) перевірка правильності результатів, які отримані за допомогою калькулятору розмитнення.

Для того аби отримати бажаний результат маю поставити вимоги і вони будуть наступними:

а) проаналізувати, як Selenium використовується в новітніх підходах при автоматизації тестів. А саме основна націленість буде на аналіз у контексті веб-сайтів у яких присутні динамічні елементи, бо я буду тестувати калькулятор, а в ньому можуть відбуватися зміни в залежності від введених нами даних;

б) розглянути та дослідити детально які є можливості у Selenium, а також, які особливості має ця технологія. Переглянути та вивчити наявну документацію, раніше виконані тести та всю інформацію, яка може бути важливою при створенні власних тестів;

в) вивчити наявну інформацію по обраному веб-застосунку для тестування, у моєму випадку це є калькулятор, який допомагає у обрахунку вартості розмитнення транспортних засобів за допомогою вказаної користувачем інформації;

г) налаштувати середовище для подальшої роботи та створення тестів;

д) створити тести для перевірки функціоналу калькулятору розмитнення. Тести мають перевіряти основні можливості калькулятору та перевіряти коректність роботи всіх функцій;

е) оцінити результати, які отримала під час тестування. Підбити по ним підсумки та на основі цих результатів сформувати загальний висновок про роботу калькулятору та окреслити подальші плани по роботі та вдосконалені функціональності та стабільності роботи;

ж) сформувати та виписати результати в пояснювальну записку, де детально описати всю пророблену роботу, інструменти, технології які використовувала у цій роботі. Проаналізувати ефективність проведеного автоматизованого тестування, провести порівняння тестів та витраченого мною ресурсу для виявлення дефектів та покриття основних наборів тестування. А також, сформувати подальші рекомендації.

2 АНАЛІЗ І ВИБІР ІНСТРУМЕНТІВ ТЕСТУВАННЯ

2.1 Характеристика Selenium: його компоненти та архітектура

Загалом, якщо досліджувати автоматизоване тестування, то можна сказати, що це дуже затратна справа і її основна задача є виконання рутинного тестування за людину, тобто те що потребує монотонних дій з боку тестувальника, наприклад, це може бути натискання певних кнопок, заповнення спеціальних форм або ж працювати з динамічними елементами. Для таких цілей і підходить Selenium WebDriver, бо він зможе зменшити навантаження на робітника та спростити загальний процес перевірки.

Загалом, структура Selenium є логічною, вона складається з певних логічних модулів і кожен з цих модулів відповідає за окрему функцію, як наприклад, Selenium WebDriver – найголовніший інструмент, за допомогою якого напряму відбувається контроль браузеру. Він працює так, що до браузера надходять команди, які йому передає тестовий скипт. Також, можна віднести сюди і бліотеки мов програмування – у Selenium є можливість використовувавти API для обраних тестувальником мов, а це є значною перевагою, бо до будь-якого проекту, навіть існуючого можна додати автоматизовані тести і не буде проблем з інтеграцією. Можна використовувати різні типи локаторів і за допомогою них виконувати обрані операції. До структру також входить Browser Drivers – це сервери, які регулюють роботу WebDriver та браузера. Ці драйвера необхідно встановлювати спеціально для кожного браузеру, наприклад, для Google Chrome використовуємо ChromeDriver і так можна підібрати для кожного браузеру. І звичайно ж, Selenium Grid – це компонент за допомогою якого можна запустити паралельне тестування використовуючи при цьому різні браузери або ж платформи. За допомогою цього можна одразу перевірити велику кількість комбінацій [2].

Розглянемо більш детально архітектуру Selenium, побудова цієї архітектури основана на клієнт-серверній моделі, саме завдяки цій побудові вона має можливість гнучко працювати та запускати тести у будь-якому браузері та середовищі і при цьому не змінювати логіку в коді (рисунок 2.1).

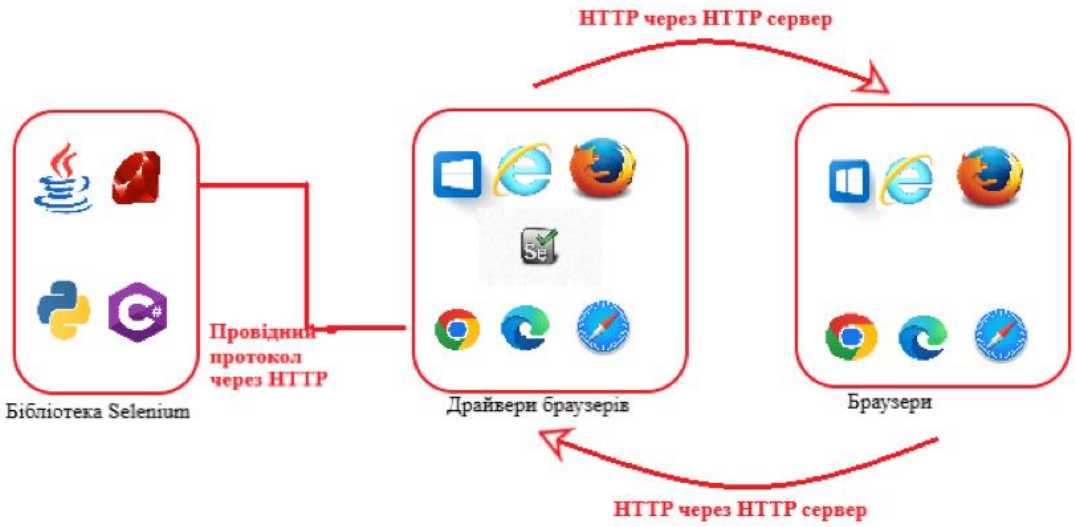

Рисунок 2.1 – Архітектура Selenium

Як видно на рисунку 2.1, то архітектуру Selenium можемо поділити на три рівні.

Клієнт – це рівень який відповідає за розташування бібліотеки Selenium. Інтегрувати можна на обрану мову програмування, наприклад, Java, Python, C#, Ruby. На цьому рівні ми формуємо сценарії для тестів, в тестах описуємо покроковість дій користувачів. Кожен крок це як певний запит HTTP, який він надсилає браузеру, команди надсилаються через WebDriver API.

Посередник – він напряму приймає запит від попереднього кроку, а саме бібліотеки Selenium і далі перетворює їх на низькорівневі команди, саме ці команди вже є зрозумілі та доступні браузеру.

Виконавець – третій блок, це вже є фінальний етап де і проходить тест.

Виконавець приймає команди, які надсилає браузер та виконує їх, а далі надсилає результат з відповіддю назад в початковий тестовий код.

Загалом, вся архітектура працює з WebDriver протоколом, а він грунтується на HTTP, бо для коректного виконання команди, наприклад, кліку необхідно надіслати запит HTTP до драйверу. За допомогою цього можна запускати навіть віддалено. Архітектура Selenium є надійною та цілком безпечною, бо як бачимо на картинці кожен блок ізольований один від одного і не має прямого доступу до коду, це зменшує ризик випадкового вторгнення у процеси браузера, а також допомагає запобігти конфлікту, який може виникнути між кодом та браузером, а отже таку структуру можна використовувати при тестуванні великих проектів [2].

2.2 Вибір мови програмування та налаштування середовища

При підготовці до створення тестів одним з найважливіших етапів є обрати правильну мову програмування на якій буде в подальшому комфортно працювати. Тому, для розгляду було обрано декілька мов, які підтримує Selenium WebDriver : Java, C#, Python, Ruby, Node.js. Для своєї кваліфікаційної роботи я обрала саме Java, бо це мова програмування, яка є найбільш популярною та має багато переваг, через що часто використовується в комерційних цілях. Також, Java це найпоширеніша мова для автоматизації та написання тестів, бо вона є надійною та стабільною, має безліч бібліотек які можна використовувати для роботи, а також дозволяє керувати залежностями та інтегрувати тести в CI/CD – процеси, а ще Java підтримує найпопулярніші фреймворки.

Якщо порівнювати з іншими популярними мовами, то в них є більше недоліків , наприклад, хоч у Python є простий та зрозумілий синтаксис і він може підійти навіть тестувальникам які лише починають свій шлах в автоматизації, але в нього гірша структура порівняно з Java, а тому буде складніше використовувати у маштабних проектах. C# також є гарним

варіантом, проте є не наскільки поширеним коли мова йде про тестуання веб-застосунків, те ж саме можна сказати і про Ruby, бо він має лаконічний синтаксис, але зараз вже втрачає свою актуальність і все менше використовується розробниками та тестувальниками при написанні тестів. Node.js же порівняно з Java менш стабільний, а також, через те що має менше документації по навчанню для роботи з Selenium, то це може ускладнювати освоєння матеріалу для подальшої роботи [3].

Отже, після порівняння можна зробити висновок, що мій вибір мови програмування є найбільш вдалим та обгрунтованим, бо в Java поєднуються всі необхідні інструменти для повноцінної роботи та обслуговування в подальшому тестів. А також вона є надійною та зручною для повноцінного створення фреймворку для автоматизації тестів для тестування веб-застосунків за допомогою Selenium.

Для того аби почати працювати з Selenium потрібно спочатку налаштувати середовище в якому буде проводитися робота. В моєму випадку для роботи я обрала мову програмування Java, а для того аби все вірно працювало необхідно встановити спеціальні компоненти, які дозволять напряму писати тести і далі з ними працювати, а також налагоджувати тести.

Для початку треба встановити JDK, рекомендую обирати саме цей варіант, бо це є офіційний набір інструментів, який працює з Java, а також, є сумісним з Selenium WebDriver та підтримує всі необхідні бібліотеки.

Наступним кроком встановлюю Apache Maven – це спеціальна система, яка дозволить керувати проектами та збірками, які в подальшому буду використовувати в автоматизації, компіляції таестів і тд.

Для коректної роботи необхідно також налаштувати параметри середовища та прописати шляхи для JDK, Maven, IntelliJ IDEA (рисунок 2.2).

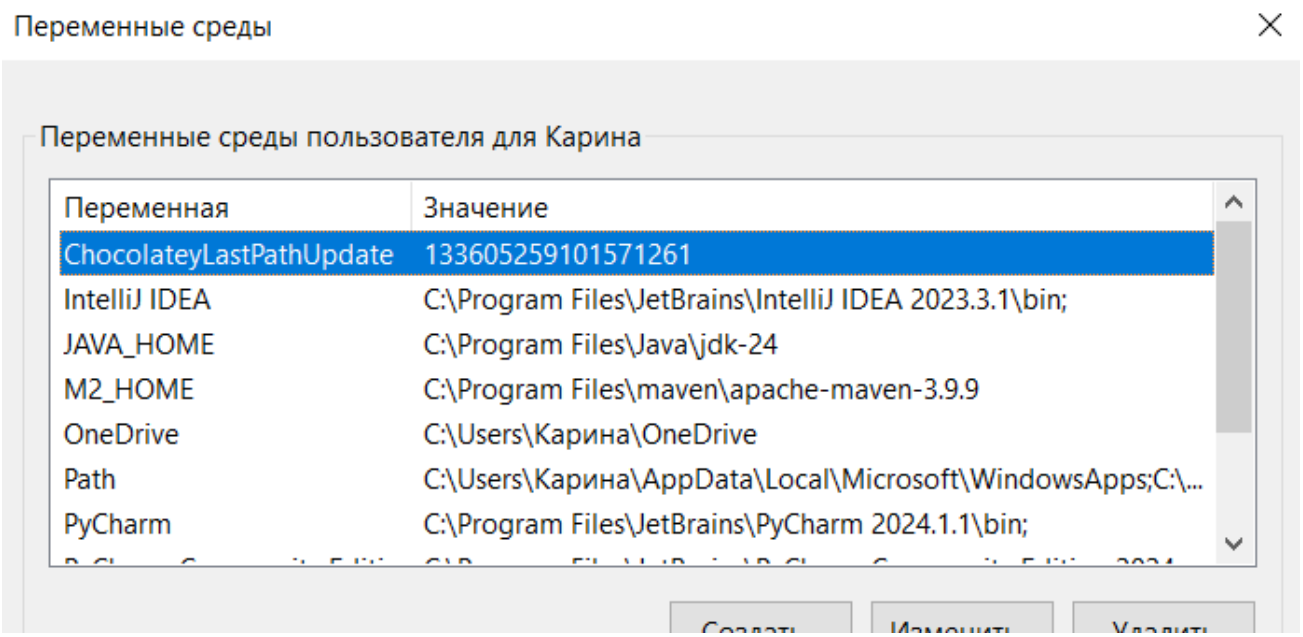

Рисунок 2.2 – Додавання шляхів для JDK, Maven, IntelliJ IDEA

Після встановлення всіх необхідних компонентів та прописання шляхів можемо перейти в IntelliJ IDEA та створити перший проект (рисунок 2.3).

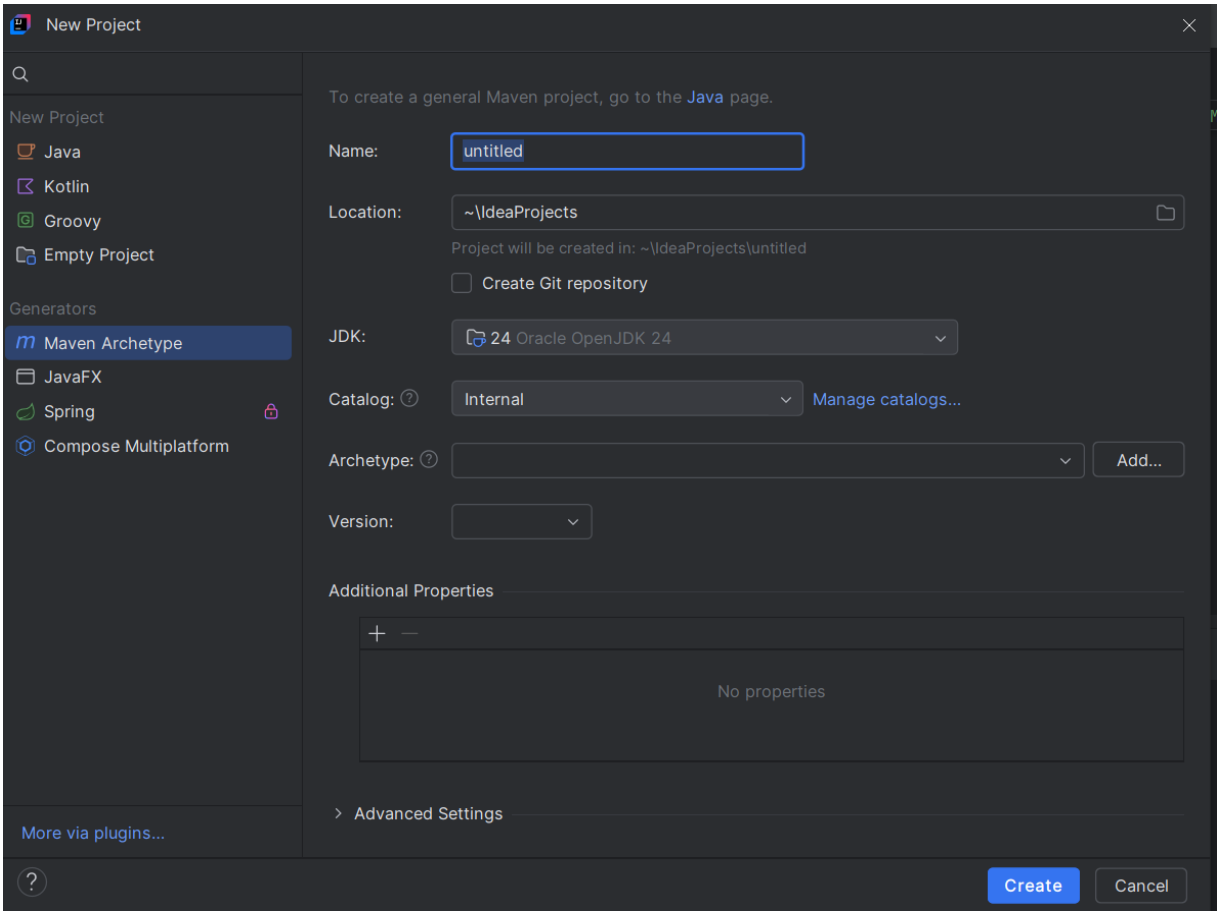

Рисунок 2.3 – Створення першого проекту в IntelliJ IDEA

Для створення проекту тестувальнику перш за все необхідно обрати платформу за допомогою якої буде відбуватися в подальшому автоматизація, обираємо Maven, так як він володіє перевагами та має чітку структуру проекту, яка є зрозумілою і з нею можуть впоратись навіть новачки, також перевагою є спрощена інтеграція, автоматичне керування наявними залежностями, а також гнучкість та маштабованість. Але, окрім обраного мною Maven можна скористатися і іншими варіантами, які також підтримуються, наприклад Apache Ant, SBT (Scala Build Tool), Gradle, Groovy [6].

Для подальшої роботи з офіційного сайту користувачу необхідно напряму завантажити Selenium, а також завантажити WebDriver для браузеру, у моєму випадку, було обрано ChromeDriver, який я завантажила з сайту - http://chromedriver.chromium.org/downloads, також користувач може обрати для себе у якості браузера який буде використовувати – Mozilla Firefox, Microsoft Edge, Safari або інші і завантажити для них відповідні драйвера.

Після встановлення необхідного драйвера користувачу необхідно його додати до проекту, для цього в папці «Test» створюємо папку «resources» (рисунок 2.4) [5].

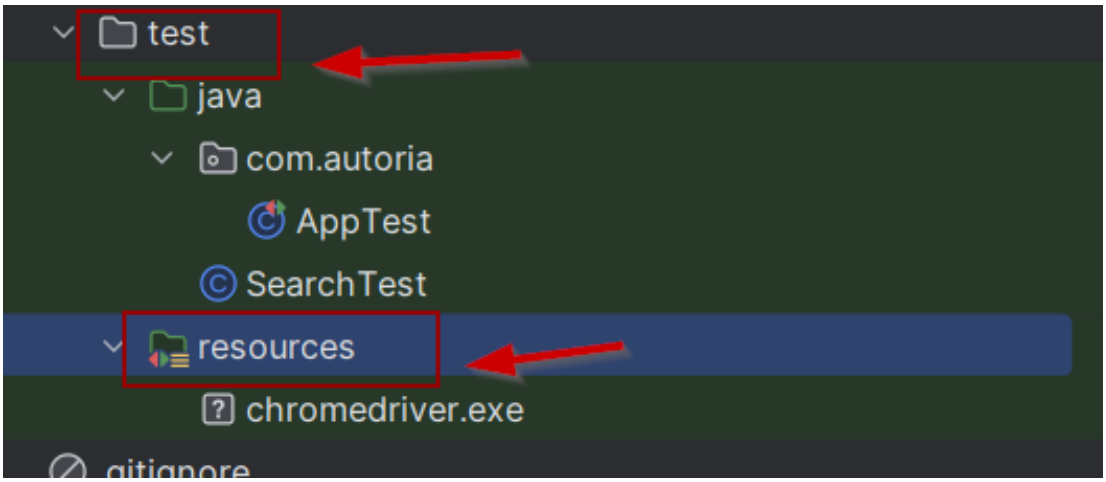

Рисунок 2.4 – Створення окремої необхідної папки для додавання залежностей

Наступним кроком буде безпосереднє додавання веб-драйвера браузера, який раніше заздалегідь встановили, це є обов'язковою умовою, так як без неї в подальшому неможливо буде провести автоматизацію тестування, бо тестування напряму здійснюватиметься у обраному браузері.

Тому, знаходжу необхідний драйвер «chromedriver.exe» та шляхом копіювання додаю в створену папку (рисунок 2.5).

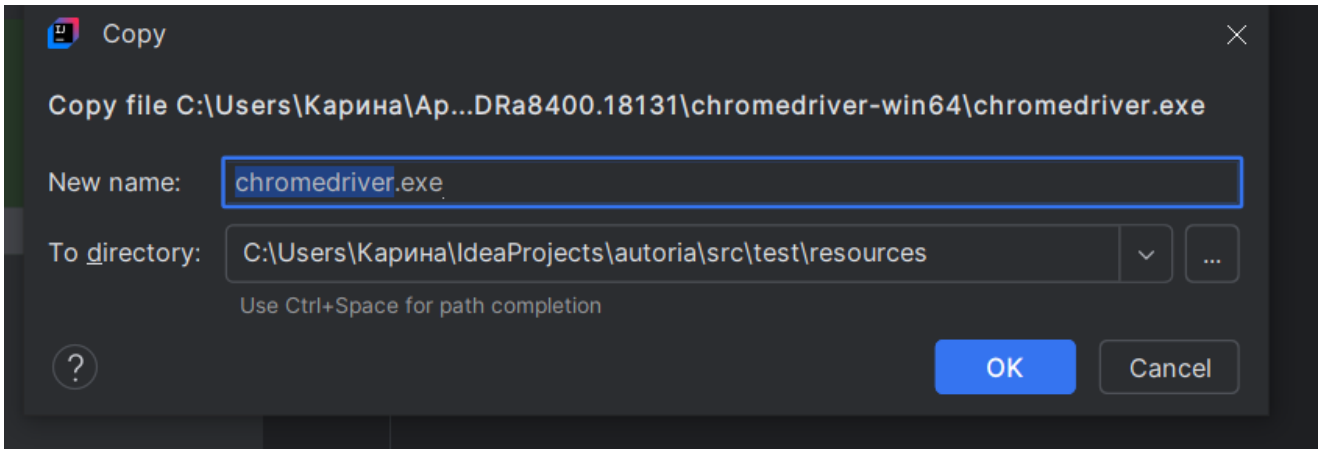

Рисунок 2.5 – Додавання драйверу «chromedriver.exe»

Після цього тестувальнику необхідно знайти всі необхідні залежності та додати їх до проекту, так як він створений на платформі Maven, то пошук залежностей найбільш оптимально буде виконувати за допомогою сайту - https://mvnrepository.com, адже на ньому можна знайти код для всіх найпопулярніших залежностей, для додавання в проект можна скопіювати вже готові скрипти та перевірити після цього як це буде виглядати в проекті (рисунок 2.6) .

До свого проекту додаю наступні залежності Selenium Java версії 4.30.0 (лістинг 2.1).

Лістинг 2.1 – Додавання залежностей до проекту

```
<!--
https://mvnrepository.com/artifact/org.seleniumhq.selenium/selen
ium-java -->
<dependency>
  <groupId>org.seleniumhq.selenium</groupId>
```

```
  <artifactId>selenium-java</artifactId>
  <version>4.30.0</version>
</dependency>
TestNG версії 7.1.0:
<!-- https://mvnrepository.com/artifact/org.testng/testng -->
<dependency>
  <groupId>org.testng</groupId>
  <artifactId>testng</artifactId>
  <version>7.11.0</version>
  <scope>test</scope>
</dependency>
```

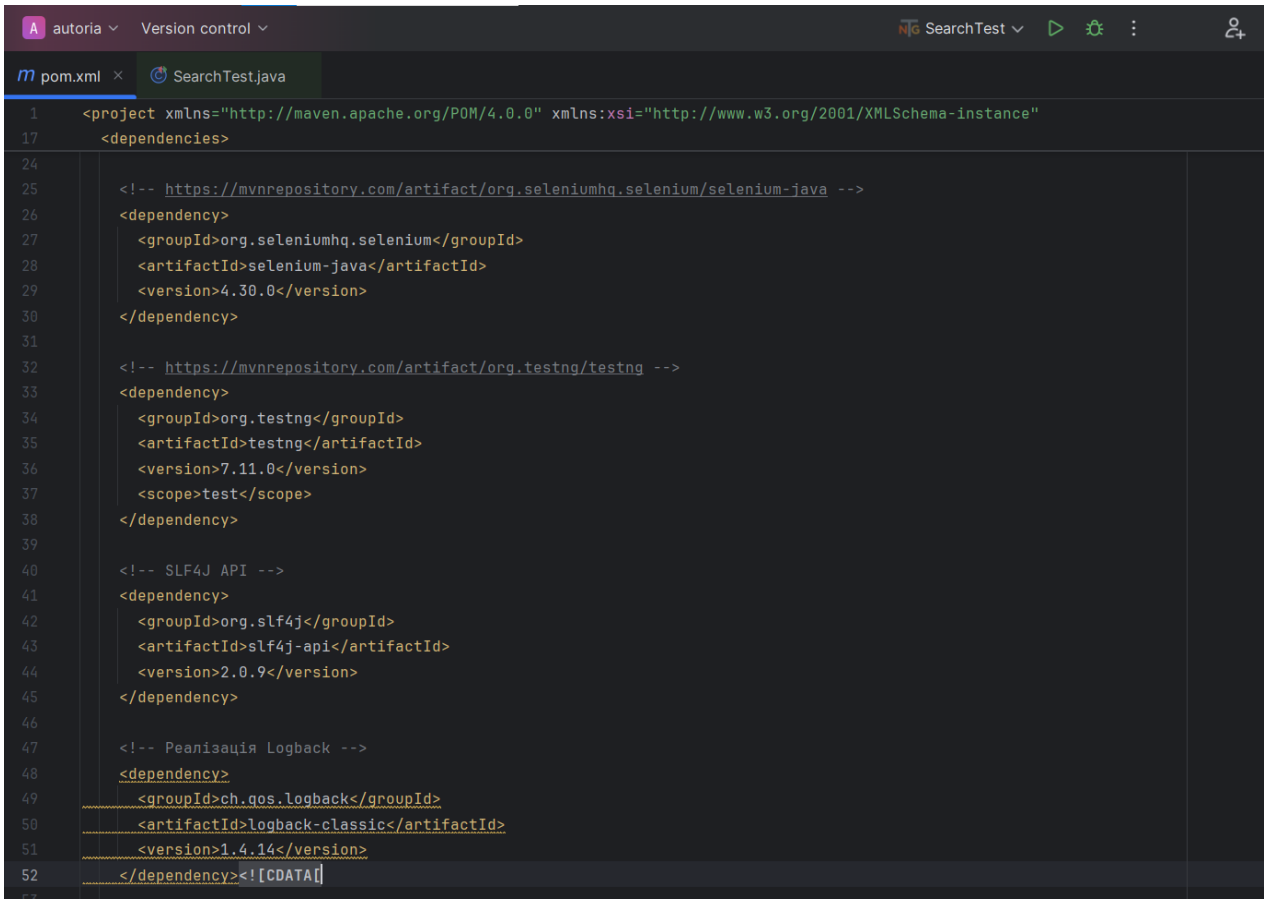

Рисунок 2.6 – Додані скрипти залежностей в проект

Тепер все готове для створення тестів, тому можливо вже писати перший тест та перевіряти коректність його роботи (рисунок 2.7).

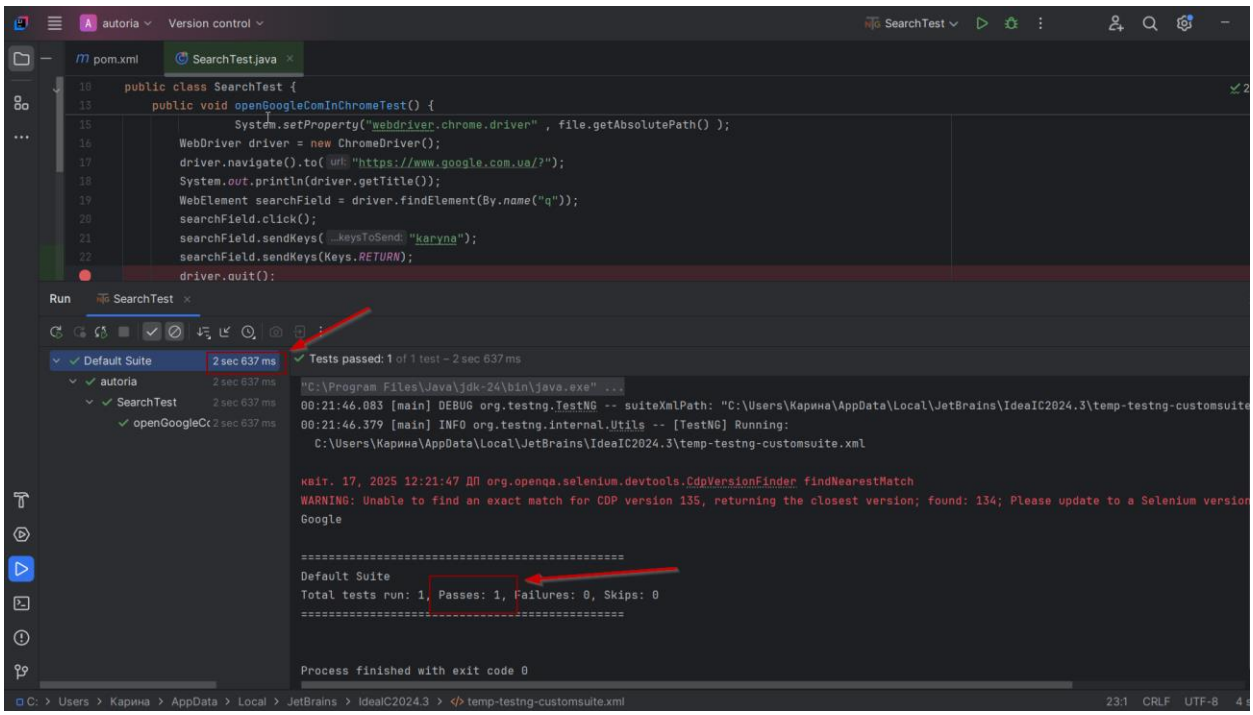

Рисунок 2.7 – Результат виконання першого пробного тесту

Отже, на рисунку 2.7 бачимо, що тест було виконано успішно. Даний тест-кейс перевірив відкриття сторінки Google, введення в пошукову стрічку запиту та пошук, як бачимо немає ніяких помилок або ж збоїв під час виконання тесту. На рисунку видно, що тест виконувався 2 секунди 637 мілісекунд, а зелені галочки вказують, що всі тести було виконано успішно, тому робимо висновок, що всі налаштування виконано успішно та можна переходити для подальшої роботи.

3 ОПИС ВЕБ-ЗАСТОСУНКУ, ЩО ТЕСТУЄТЬСЯ ТА ВИЗНАЧЕННЯ ФУНКЦІОНАЛУ

3.1 Призначення веб-застосунку та характеристика

Для написання кваліфікаційної роботи та тестування з використанням Selenium IDE мною було обрано «Калькулятор розмитнення» : https://auto.ria.com/uk/rastamozhka-avto/calculator/ , він дозволяє виконати розрахунки у вартості розмитнення авто пригнаного з-за кордону, враховучи всі необхідні платежі (рисунок 3.1).

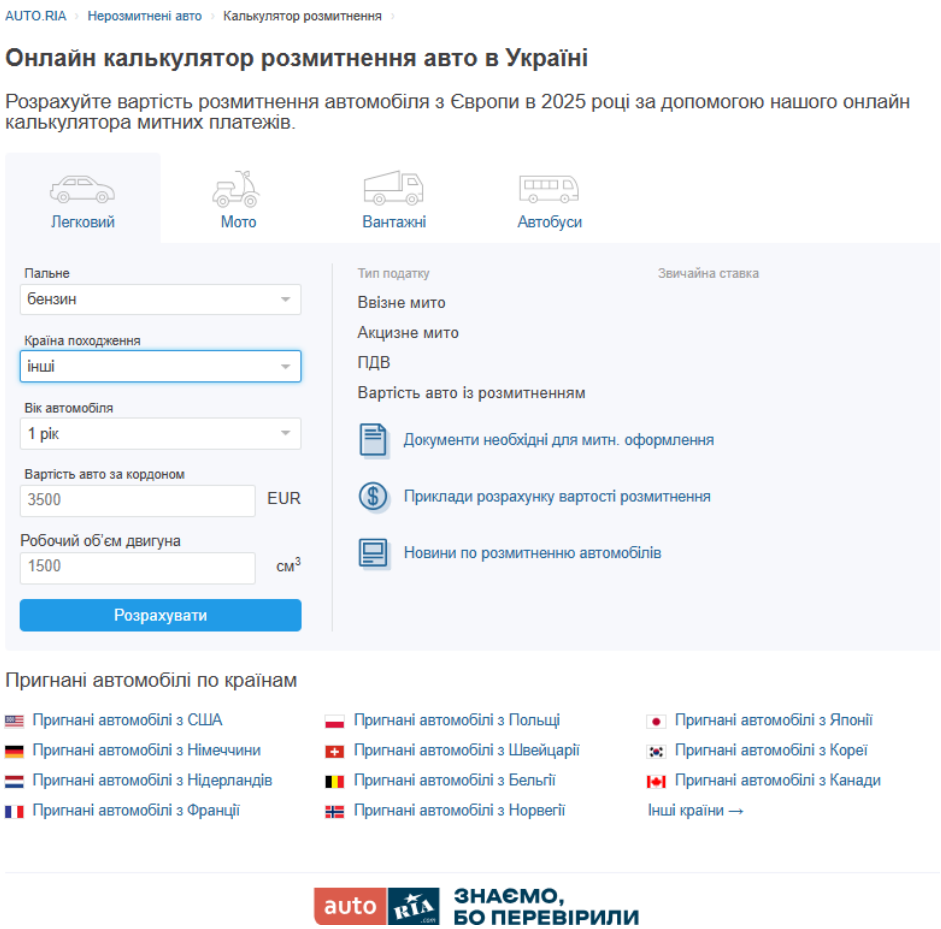

Рисунок 3.1 – Вигляд інтерфейсу веб-застосунку, що тестується

Митне оформлення – це процес, який допомагає офіційно легалізувти

ввезення авто на території України з відповідністю чинному законодавству. А результатом оформлення є сплата призначених обов'язковив податків людиною, яка ввозить авто, після чого буде можливість отримати документи та перейти ло наступного етапу реєстрації авто.

Судячи з інтерфейсу бачимо, що основним завданням веб-застосунку є забезпечення швидкого механізму по розрахуну митних платежів, для того аби підрахунки були вірними користувачу необхідно знати та вказати наступне:

а) тип палива у авто;

б) з якої країни ввезений автомобіль;

в) вік авто;

г) вартість за яку придбали авто за кордоном;

д) робочий об'єм двигуна.

Після введеня всіх цих даних користувач в результаті отримає розрахунки по: ввізному миту, акцизному миту, податку на додану вартість(ПДВ) та в кінці загальну суму розмитнення, яка і є фінальною вартістю розмитнення (рисунок 3.2).

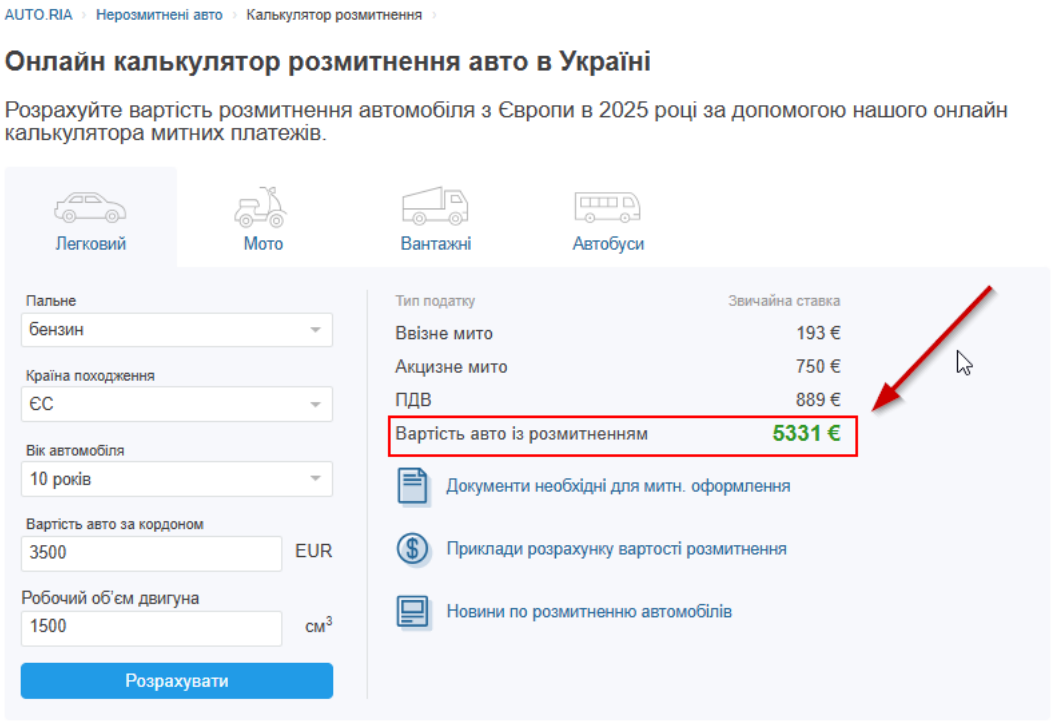

Рисунок 3.2 –Приклад розрахуку за допомогою калькулятору

Отже, як бачимо , що фінальна сума за вартість авто із розмитненням буде залежати від трьох основних факторів:

а) акцизний збір: ця форма податку залежить типу транспортного засобу, типу двигуна авто, наприклад, дизельні двигуни,бензинові, електро, також від об'єму двигуна та віку авто. Розрахувати збір можна за формулою: Акциз = $Ставка^{база} * К^{вік\ авто} * К^{об'єм\ двигуна}$;

б) для електричних авто податок дорівнює 1 євро на 1 кіловат/годину електро акумулятора для авто;

в) Для авто з гібридним типом двигуна податок становить 100 євро за 1 штуку;

г) $ставка^{база}$ податку оцінюється в євро за 1 транспортий засіб і напряму залежиь від двигуна, так наприклад у дизельних двигунів є розділення до 3500 $см^3$ та понад. Для першого варіанту, де об'єм менше 3500 $см^3$ (включно) = 50 євро, для другого з об'ємом понад 3500 $см^3$ = 100 євро. Для двигунів з бензиновим типом палива та об'ємом до 3000 $см^3$ (включно) = 50 євро, а для тих де об'єм більше 3000 $см^3$ = 100 євро;

д) $К^{вік\ авто}$ - це коефіцієнт, який напряму залежить від кількості років авто(повних календарних). Якщо транспортний засіб був у вжитку до одного включно календарного року, то коефіцієнт =1, для авто які старше 15 років = 15;

е) $К^{об'єм\ двигуна}$ - розраховується за формулою ділення об'єму циліндру двигуна $см^3$ на 1000 $см^3$;

ж) ввізне мито: це мито дорівнює 10% від зазначеної вартості автомобіля, а для електричних авто 0%;

з) податок на додану вартість: ПДВ дорівнює 20% від зазначеної вартості, але важливим є те, що вона не має бути нижчою за митну вартість, в яку входять мито, акцизний податок, які підпадають під сплату та включаються до вартості авто [4].

3.2 Структура і основні компоненти

Калькулятор розмитнення можна рахувати як інтерактивний інструмен щоб швидко та максимально точно отримати рахунок по митним платежам для обраного авто. Загалом, структура даного веб-завтосунку об’єднує в собі логічні модулі, які пов’язані між собою загальними правилами, але при цьому кожен виконує окрему функцію і займає свою роль у процесі розрахунку та виведені фінальної суми розрахунку. В ньому закладено логіку для обчислення, перевіряє валідацію даних, які вводить користувач, ну і власне видає готовий результат. Переважна більшість кроків, які виконують користувачі приводиться до того, що йому треба ввести дані у спеціальні поля за допомогою чисел, а також, можливості обирати запропоновані параметри серед наявних.

Першим компонентом структури є – інтерфейс взаємодії з корисувачем, де відбувається збір інформації про авто для якого бажають отримати інформацію про розмитнення. Для того аби було зручно користуватися калькулятором розмитнення користувачу були залишені підказки, наприклад, які типи пального є, а також з якої країни ввезено. Там де треба ввести числові параметри, то там також є підказки у вигляді в якій метриці відбувається вимірювання двиуна або ж валюту. Це зроблено для того аби користувач міг все вірно ввести, бо якщо буде допущено похибку, то це напряму вплине на фінальний результат у розрахунках, бо до кожного типу застосовуються свої ставки для оподаткування і тд.

Другим компонентом структури є не менш важливий модуль, який відповідає за обробку даних, які ввів користувач. Основним його завданням є перевірити чи вірно ввів користувач данні, чи не було пропусків в полях. Наприклад, якщо пропущено поле з введенням пального або ж об’єм двигуна. Якщо щось буде пропущено або ж введено не вірно, то користувач отримає повідомлення про помилку у введенні, чи про пропуск.

Одним з найважливіших компонентів у структурі є сам модуль який

буде рахувати суму митних платежів. Він вже налаштований на основі математичних формул та обчислень,відсотків, які оновлюються згідно онвленнюю в умовах розмитнення, актуальні оновлення ж можна переглянути на сайті https://customs.gov.ua/mitne-oformlennia-avtomobilia. Формули для кожного з компонентів було вказано вище в пункті 3.1, а також там же показано приклади їх застосування.

Після цього спрацьовує фінальний модуль, який покаже користувачу готовий результат і клієнт може його побачити також в структурованій подачі. Цей результат опрацьовується за допомогою модулю відображення результатів.

Якщо підбивати підсумки, то у калькулятора розмитнення є чітка та логічна структура, яка допомогає отримати вірний фінальний результат і при цьому не є складною для розуміння користувачів. Вигляд, який відображає послідовність дій в роботі (рисунок 3.3).

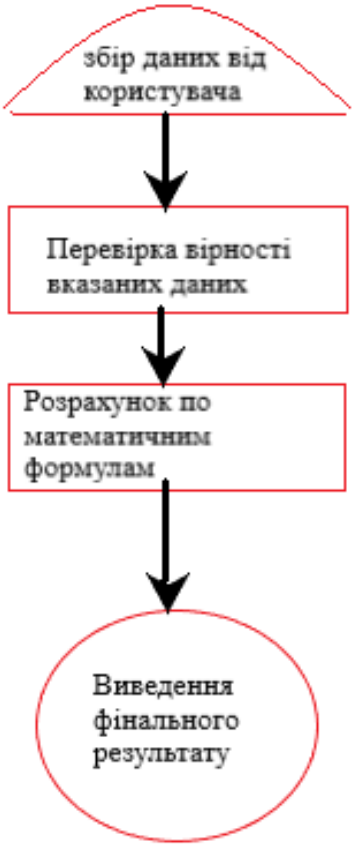

Рисунок 3.3 – Схема послідовності дій при обчисленні результатів митного оформлення

Саме за допомогою такої архітектури та наявних модулів забезпечується безперебійна робота, а також швидкість обчислень, стабільності та зручний формат отримання результатів для користувачів.

4 АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ З ВИКОРИСТАННЯМ SELENIUM

4.1 Визначення функціональних вимог до тестування

За допомогою функціональних вимог визначається поведінка системи при взаємодії з юзером, а також, реакцію системи на введені дані. У моєму випадку у калькулятора розмитнення авто головною метою при проведенні тестування є перевірка точності обчислень при розрахунку митних платежів, а також у вірності валідації вхідних даних, правильне реагування системи при зміні користувачем вхідних даних, а саме оновлення результатів у фінальній відповіді.

Отже, функціональні вимоги до веб-застосунку калькулятора розмитнення:

а) вимоги до обчислення

1) розрахунок митних платежів:

1.1.1 митний платіж має бути розрахований як відсоток від вказаної вартості авто;

1.1.2 ставка для країн ЄС має дорівнювати 10%, а у випадку електромобілів 0%;

1.1.3 митний розрахунок має виводитись у прийнятій валюті – євро .

2) розрахунок акцизного збору:

1.2.1 система має автоматично розраховувати акцизний збір згідно формулі : $Ставка^{база} * К^{вік\ авто} * К^{об'єм\ двигуна}$;

1.2.2 фінальне обчислення має відображатись в окремій колонці для відображення результатів розрахунку;

1.2.3 при неправильному внесенні даних система не має розраховувати результат та має видати помилку, підсвітивши користувачу відповідне поле, де треба скорегувати дані.

3) розрахунок ПДВ:

1.3.1 фінальна сума ПДВ має становити 20% від загальної суми;

1.3.2 результат виводиться у відповідній колонці, сума прописується в євро.

4) загальна сума розмитнення:

1.4.1 фінальна сума розмитнення має розраховуватись за формулою: Сума мита + вартість акцизу + ставка ПДВ;

1.4.2 остаточна сума має відображатися валюті євро, як було і в попередніх розрахунках;

1.4.3 підсвічується зеленим кольором та виводиться жирним шрифтом у фінальній стрічці розрахунків.

б) вимоги до інтерфейсу виведення результатів:

1) відображення результатів розрахунку:

2.1.1 кожен з розрахованих податків має відображатись в окремій колонці та має бути підписаним відповідно до розрахунку;

2.1.2 біля кожного обчислення має бути прописана валюта в якій проводився розрахунок(у нашому випадку євро);

2.1.3 фінальна сума розмитнення має бути виокремлена від інших та бути інтуітивно зрозумілою для користувача за рахунок підсвічення окремим кольором та виділення жирним шрифтом.

в) вимоги до інтерфейсу:

1) першочергове відображення полів:

3.1.1 система повинна відображати загальні поля, які будуть доступні для вводу, а саме: пальне, країна, вік авто, вартість авто за кордоном, об’єм двигуна;

3.1.2 має бути доступна можливість вибору типу транспорту.

2) Валідація полів для введення значень:

3.2.1 поля приймають лише додатні значення та неможливо ввести негативне значення;

3.2.2 система не має приймати в поля вводу буквені, чи символьні значення;

3.2.3 при введенні не вірних користувач має отримати сповіщення про помилку та в якому блоці вона сталась;

3.2.4 при виборі типу палива – електро, має приховатись поле з об'ємом двигуна авто та змінитись на об'єм АКБ, а також прибратись поле вік авто, бо це є не важливий параметр при розрахунку;

3.2.5 при зміні типу транспорту мають оновлюватися поля, які будуть підлаштовуватися під обраний тиа транспорту;

3.2.6 зміна будь-якого параметру має автоматично скидати попередні параметри обчислення та пропонувати користувачу ввести нові дані для обчислення.

г) поведінкові вимоги:

1) адаптивність інтерфейсу користувача:

4.1.1 калькулятор розмитнення має бути доступним та коректно відображатись на різних розширеннях екрану.

2) якщо користувач спробує розрахувати без заповнення одного або декількох полів відповідними параметрами, то система не дасть цього зробити та видасть помилку.

д) додаткові вимоги, які можна розглянути для калькулятора розмитнення:

1) всі розрахунки мають проводитись на основі формул та законів, які можна переглянути на сайті: https://customs.gov.ua/mitne-oformlennia-avtomobilia;

2) з калькулятора мають бути доступні додаткові посилання на інформацію про розмитнення та додаткові джерела про документи необхідні для митного оформлення та інша корисна інформація, яка допоможе користувачу;

3) усі написи та результати обчислень мають бути локалізованими.

4.2 Формування тестових сценаріїв

Тестовий сценарій – це загальні описи, які використовуються для тестування, за допомогою них визначається як саме має виконуватися тестування для обраного тестувальником компонента в системі або ж якоїсь окремої функції.

Для того аби протестувати калькулятор розмитнення на AUTO.RIA потрібно перевірити коректність роботи веб-застосунку, потрібно опиратися на функціональні вимоги, які створювалися раніше. Спочатку необхідно перевірити чи вірно відображаються всі елементи сторінки в інтерфейсі користувача при першочерговому завантаженні, тому саме на це і буде спрямований перший тест. В цьому тесті основні перевірки будуть зосереджені на початковій формі, тобто має бути перевірка параметрів за замовчуванням, наприклад, випадаючі списки, пусті поля для введення даних. Також, перевірка буде в себе включати відображення області з результатами фінального розрахунку, а також те, що не відображається жодного результату якщо дані не введені або ж введені некоректно.

Наступним кроком буде створення тестового сценарію який допоможе перевіряти акцизний податок для авто, з урахуванням даних введениз про авто. Результати розрахунку мають співпадати з загальноприйнятою офіційною формулою, в якій закладені коефіцієнти для параметрів авто.

При створенні тестових сценарієм є дуже важлива перевірка на взаємозв’язки між полями і значенням, які вводить користувач в ці поля. Після того як користувач введе всі необхідні дані в поля і натисне кнопку розрахунку, то вони мають відповідати фінальному розрахунку та підпадати під умови математичних формул.

Додатково також може бути розроблено сценарії для електромобілів, гібридних авто, які можна зробити також за аналогією з попередніми прикладами тестових сценаріїв. Всі створені тестувальником сценаріх можна об’єднати в загальний спільний для них тестовий набір і за допомогою цього

автоматизовано запустити готові серії тестів для того аби перевірити цілісну роботу калькулятора. Саме такий варіант тестування значно спрощує процес тестування при регресійному тестуванні, що дозволяє швидко перевірити основну функціональність після того як розробниками було внесено зміни в код або ж виправлено певний баг.

4.3 Налаштування середовища Selenium IDE

При сучаному тестуванні все частіше постає пореба у надіних і при цьому ефективних інструментів, які будуть допомагати тестувальникам у виконанні роботи та забезпечені стабільності та безеребійності роботи. Наразі, одним з найпопулярніших таких рішень виступає Selenium IDE, бо за допомогою нього можна виконувати одразу багато функцій, а саме дозволяє редагувати, записувати та напряму виконувати тести, при цьому це підходить і для тестувальників, які поки не володдіють високим рінем у програмуванні.

Selenium IDE – це потужний інструмент плагін, який використовують в автоматизації веб-застосунків. Selenium IDE працює у вигяді розширення для браузерів Chrome, Firefox, Edge. Його можна використовувати при написанні простих скриптів, експортовувати тести у різні мови програмування, наприклад Python, Java, C# та інші. Також, він має ряд переваг, бо простий у використанні, розширення підтримується найпопулярнішими браузерами, підтримує розширений функціонал та підтримує плагін, можливість у використанні Selenium WebDriver, також, мається параметризація тестів у тестуванні. Переглянути основну інформацію та детальну документацію по Selenium IDE (https://www.selenium.dev/downloads/).

У моєму випадку було обрано для роботи браузер Chrome, тому завантажую плагін за посиланням (https://www.selenium.dev/selenium-ide/).

Після того як було встановлено розширення для браузеру Google Chrome, який забезпечив інтеграцію напряму у середовище веб- переглядача, то з’являється спеціальне розширення в кутку браузера на панелі

інструментів (рисунок 4.1).

Рисунок 4.1 – Встановлене розширення Selenium IDE

Після встановлення необхідно натиснути на розширення Selenium IDE, де відкривається першопочаткове вікно, яке дає можливість створити та записати перший проект за допомогою «Record a new test in a new project» (рисунок 4.2), після натискання користувачу необхідно вказати ім’я проекту, бажано додавати таку назву, яка описувала б суть проекту, веб-застосунку який тестується, у моєму випадку вказано згідно назві сайту, а саме «AUTO.RIA» , далі додаю URL – посилання на веб – застосунок для тестування, а саме калькулятору розмитнення, важливо щоб посилання було коректним, для того аби перехід на сторінку від самого початку був вірний, бо від цього залежить якість виконаного тесту.

Рисунок 4.2 – Першочергове вікно, яке бачить користувач після запуску

Далі для початку необхідно провести пробне тестування, для того аби перевірити коректність роботи встановленого розширення, а також, запис створеного тесту. Вхідними параметрами та діями для виконання було обране наступне:

а) Вибір типу транспорту – легкові,

б) Вибір типу палива – бензин,

в) Вибір країни походження – ЄС,

г) Вік автомобіля – 5 років,

д) Вартість авто за кордоном – 15 000 EUR,

е) Робочий об’єм двигуна – 1 700 см^3.

Після початку запису відбувається запис виконаних дій користувачем та система видає готовий результат (рисунок 4.3).

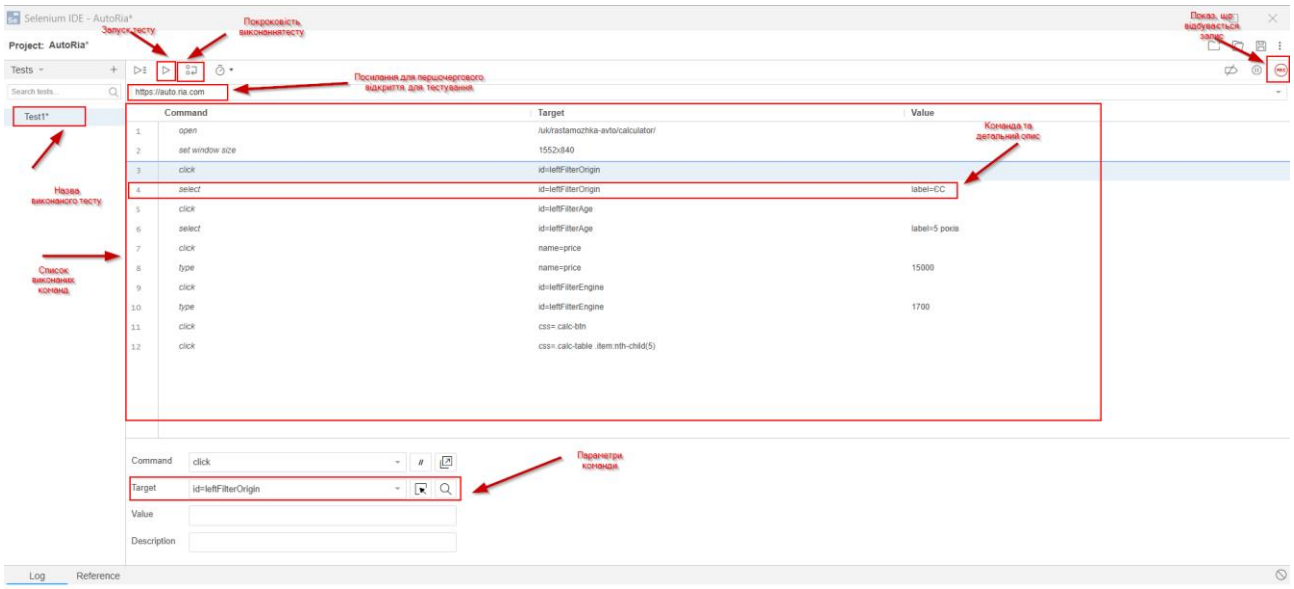

Рисунок 4.3 – Результат відображення вікна з виконаними командами для тестового сценарію

Як видно по списку виконаних команд бачимо, що перщочергово йде команда open, за допомогою неї відбулось завантаження першочергової сторінки калькулятора розмитнення. Саме ця сторінка буде відправною

точкою для початку тестування. Наступною командою було виконано «set window size», а саме це дозволили уникнути проблем з адаптивним дизайном, дозволило краще сприймати користувачу відображення, а також створити стандартні умови при тестуванні, ця команда змінила розширення на 1552 x 840 пікселів. Далі користувач перейшов до введення основних вхідних даних, за допомогою команди «select | id=leftFilterOrigin | label=ЄС». Де, можна побачити, що відбувається вибір за допомогою команди «select», далі показано випадаючий список на що вказує ідинтифікатор «id=leftFilterOrigin», та обрано варііант з позначкою «label=ЄС». Отже, в результаті бачимо, що авто імпортовано з ЄС. Наступним кроком як і в попередній команді було, де користувач з виаадаючого списку обрав вік авто за допомогою ідентифікатору «id=leftFilterAge» та позначки «label=5 років». Після цього було активовано параметр, який дозволить ввести вартість авто, а саме клікнувши на текстове поле та додавши спеціальний атрибут name=price та команди type введено вартість 15 000 EUR. Остарнім полем вводу було введено об’єм двигуна у розмірі 1700 $см^3$ за допомогою команди «type | id=leftFilterEngine | 1700». Після того як були заповненні всі параметри згідно команд видно, що користувач натиснув «Розрахувати» за допомогою команди «click» для елементу класу «calc-btn». Після чого було переглянуто детальну інформацію розрахунків, які виведені в таблиці розрахунку розмитнення за допомогою команди «click» по елементу з «css-селектором .calc-table .item:nth-child(5)».

По результату виконання тестового сценарію видно, були враховані всі особливості, як наприклад зміна розміру вікна, також реаліація показала себе з гарного боку та стабільною, бо були використані абсолютні селектори, також, видно, що усі кроки та дії користувача сгруповані згідно логіці, у ньому відсутні додаткові, чи непотрібні операції. В потальшому сценарій можна удосконалювати та додавати нові перевірки, які будуть ще більш чітко моделювати стандартні перевірки та передбачити поведінку користувача, який буде використовувати веб-застосунок для свого розрахунку і в

подальшому автоматизувати ці тести для дослідження коректності роботи в калькуляторі розмитнення.

Також, у Selenium IDE є команди за допомогою яких і виконуються дії, тому розглянемо їх більш дедально та на які типи вони поділяються:

а) Першим типом є команди Actions вони створені щоб автоматисно проходити тест, перевіряти як працює інтерфейс. Тобто створювати дії які міг би виконувати звичайний користувач і натискати на кнопки, чи вибирати певну дію зі списку (табл. 4.1).

Таблиця 4.1 – Команди типу Actions

Назва команди	Опис	Приклад
click	Клік лівою кнопкою миші на обраному елементі	Id=btn=submit
doubleClick	Двойний клік на обраному елементі	Css=.product-name
select	Можливість вибору певної опції з заданого випадаючого списку	Id = city lable=Kharkiv
sendKeys	Натискання на клавіші, наприклад, enter, tab	Id=search
check	Встановлення прапорця на обраний чекбокс	Id=agree-terms
mouseOver	Наведення курсора миші на обраний елемент	Css=.menu-item
mouseDown\ Up	Відтискання або натискання кнопки миші, при виконанні складних дій при тестуванні	Id=slider

Продовження таблиці 4.1

focus	Додавання фокуса на обраний елемент	Id = name-field
type	Введення текстового значення в поле	Name=email karyna.holovan@nure.ua
dragAndDropToObject	Перетягування елемента з одного на інший елемент	Id=item1 id=drop-zone

б) Наступним типом є команди Assertions для перевірки – ці команди можна використовувати для того аби переконатися, що веб-застосунок працює належним чином, а саме так як очікується. Тобто він порівнює отримані фактичні результати з тими, що очікувалось – якщо результати зберігаються, то тест успішний, якщо ж ні – провальний. Саме цей тип команд є найважливішими в автоматизованому тестуванню, бо сме за допомогою них відбувається перевірка коректності роботи (табл. 4.2).

Таблиця 4.2 – Команди типу Assertions

Назва команди	Опис	Приклад
assertText	Перевіряє текст в елементі(точність)	Id= status
assertValue	Перевіряє значення у полі вводу	Id=total
assertTitle	Перевіряє заголовки сторінки	Home Page
assertChecked\NotChecked	Перевіряє,що відбулося встановлення\невстановлення радіокнопки або чек-боксу.	Id= newsletter \ old-user
assertVisible\NotVisible	Перевіряє, що елемент видимий\невидимий	Css=/modal\ id=loader

Продовження таблиці 4.2

assertElementPresent\ NotPresent	Перевіряє наявність\ не наявність елементу на сторінці DOM	Id=header\popup

в) Команда типу Waits створені спеціально для уповільнення при виконанні тесту, для того щод дочекатись коли з’явиться або зміниться стан обраного елемента для веб-сторінки (табл. 4.3).

Таблиця 4.3 – Команди типу Waits

Назва команди	Опис	Приклад
waitForElementVisible\NotVisible	Очікує, коли елемент буде видимим \ зникне елемент	id=loading\ id=spinner
waitForText	Очікує допоки текст з'явиться в елементі	id=statusDone
waitForElementPresent	Очікує появи елемента у DOM	id=results
pause	Призупиняє тест на певний період часу(мілісекундах вимірюється)	pause

г) Команди для навігації Navigation commands – за допомогою них відбувається керування переїодами між сторінками, також, вони можуть впливати на відкриттч потрібного сайту, оновлення, повернення назад (табл. 4.4).

Таблиця 4.4 – Команди типу Navigation

Назва команди	Опис	Приклад
open	Відкриває потрібна посилання для сайту	/login
refresh	Оновлює поточну відкриту сторінку	refresh
goBack	Дозволяє повернутися на попередню сторінку	goBack
goForward	Якщо є наступна сторінка, то переходить на неї	goForward

д) Store Commands – це команди збереження, вони працюють зі змінними,а саме зберігають значення отримані зі сторінок або ж проміжні результати і їх можна використовувати в подальшому для створення тестових сценаріїв (табл. 4.5).

Таблиця 4.5 – Команди типу Store

Назва команди	Опис	Приклад
store	Зберігається будь-яке довільне значення у змінній(число, текст)	varName
storeValue	Зберігається текст елемента	id=quantity itemsCount
storeText	Зберігає значення атрибуту елемента	id=price totalPrice

Продовження таблиці 4.5

storeAttribute	Зберігає заголовок сторінки, конкретний атрибут HTML-елемента	id=link@href
storeTitle	Зберігає значення заголовку сторінки і має свій тег <title>	pageTitle

е) Також, існують додткові команди, які можна використовувати у разі якщо треба додатково розширити тести або ж попередні команди не підійшли під потреби (табл. 4.6).

Таблиця 4.6 – Команди іншого типу(додаткові)

Назва команди	Опис	Приклад
runScript	Допомагає виконувати прописаний JavaScript-код	runScript window.scrollTo(0, 500)
echo	Надає змогу виводити значення у консоль, при дебагінгу	echo
executeAsyncScript	Виконує асинхронний JavaScript	executeAsyncScript callback()
close	Закриває відкрите(поточне) вікно в браузері	close

Вище було наведено основні і найчастіше використовувані команди, але кожен тестувальник у разі необхідності додаткової інформації може ознайомитись на офіційному сайті за посиланням:

https://www.selenium.dev/selenium-ide/docs/en/api/commands.

Не менш важливим під час проведення тестування є поле Target у Selenium IDE, бо це ключовий елемент який допомагає у ідентифікації об’єктів веб-астосунку під час автоматизованого тестування. Саме воно визначає, як саме має віднайтись елемент сторінки, для того ави виконалась якась дія над ним [7].

Для того аби вірно створити команду, необхідно знати HTML-код елемента, який обрали на сторінці, це можна зробити натиснувши на елемент правою кнопкою миші та обрати «Переглянути код елемента» (рисунок 4.4).

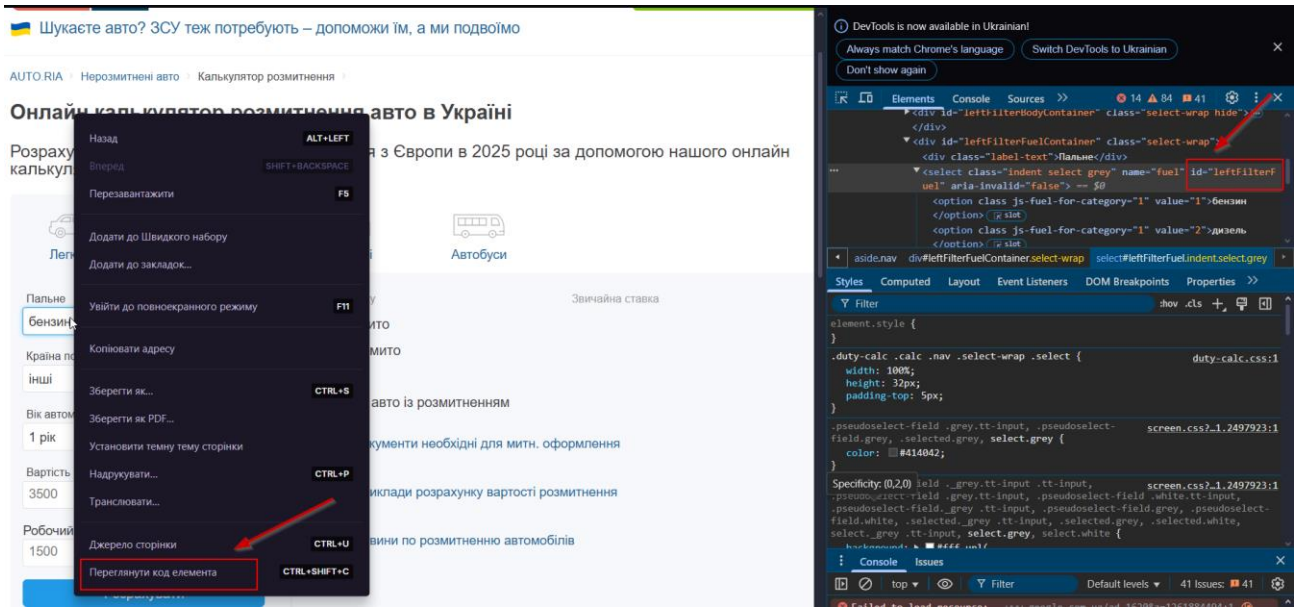

Рисунок 4.4 – Приклад знаходження необхідного елементу для подальшого тестування та перевірки в полі Target

Selenium дозволяє інтуїтивно і зрозуміло автоматично генерувати локатори та значно прискорювати створення тестів.

Для прикладу розглянемо реальне використання для перевірки та встановлення значення:

Command: «select»

Target: «id=leftFilterFuel»

Value: «label=Дизель»

За допомогою кнопки Find target in page можна знайти відповідний елемент, після того як було введено до поля Target локатор HTML – елементу, що тестується. За допомогою такої перевірки можна переконатися, що подальше тестування та конкретна команда будуть застосовані саме до бажаного елементу (рисунок 4.5).

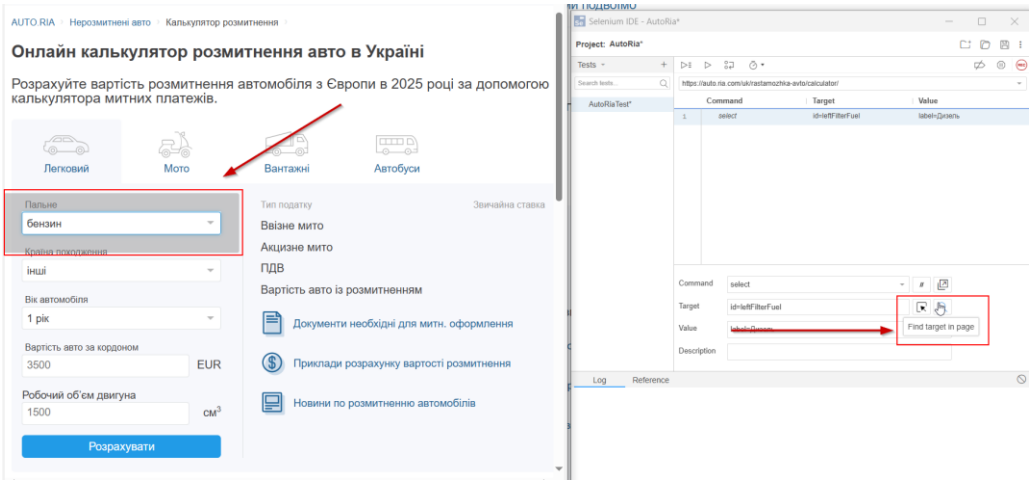

Рисунок 4.5 – Підсвічування при пошуку HTML – елемента по локатору

Також, надається можливість вибору конкретного оптимального локатору до вказаного елементу, для цього достатньо натиснути на випадаючий список у полі Target і будуть доступні інші варінти локаторів. Це зручно, бо допомагає заощаджувати час і не прописувати кожен локатор вручну, також, зменшує кількість можливих помилок при написанні синтаксису, надає варіанти на створення різних сценаріїв для тестування та можливість перевірити коректність роботи локатора (рисунок 4.6).

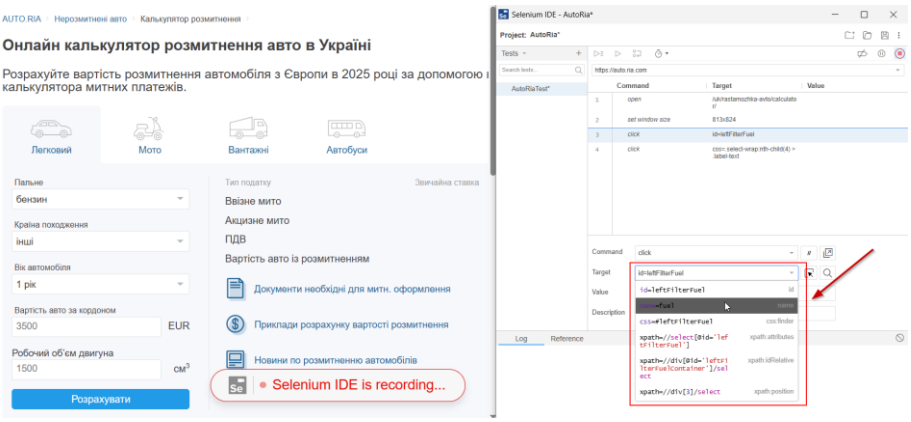

Рисунок 4.6 – Вибір додаткових локаторів HTML – елемента

Переходимо до записування тестування та проведенням основних тестів розрахунку митного розмитнення (рисунок 4.7). Для цього вмикаємо режим запису тесту за допомогою кнопки «REC» у правому верхньому кутку. Далі переходимо у браузер та заповнюємо вхідні дані, а саме:

а) Тип пального (за допомогою `id=leftFilterFuel lable = бензин`)

б) Країна походження (`id=leftFilterOrigin lable = ЄС`)

в) Вік авто (`id=leftFilterAge lable = 7 років`)

г) Прописуємо числові дані, а саме ціну (`name=price`) та об’єм двигуна (`id=leftFilterEngine`)

д) Виконуємо клік на кнопку «Розрахувати» (`css=.calc-btn`) та надсилаємо дані на сервер для розрахунку вартості розмитнення.

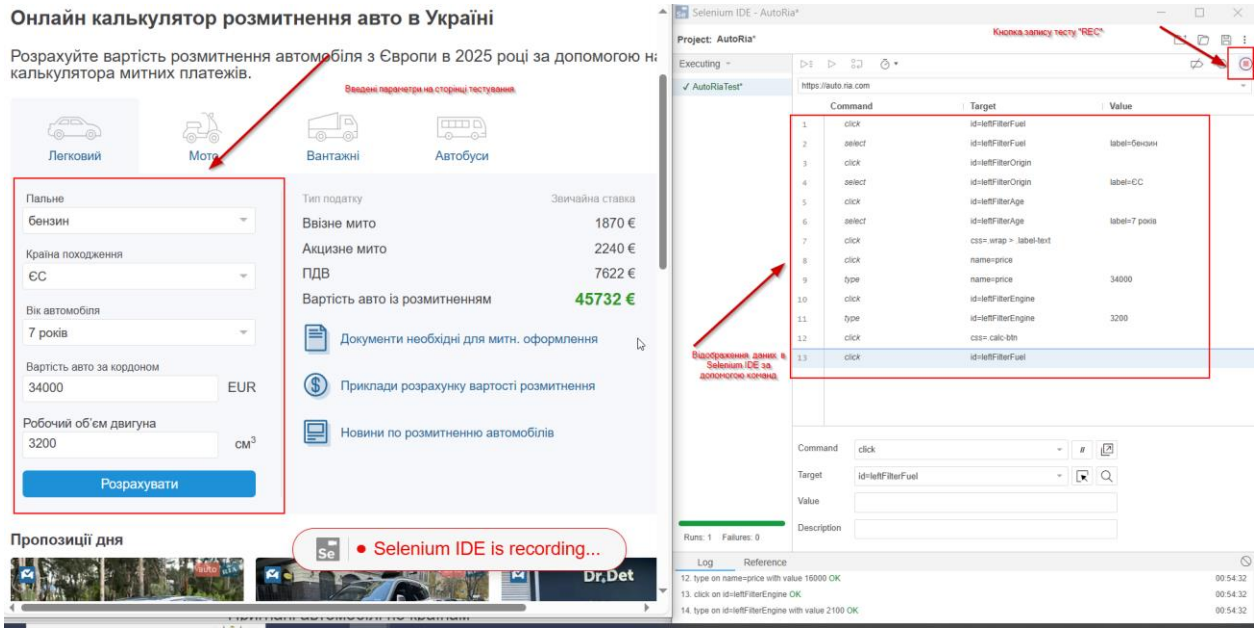

Рисунок 4.7 – Відображення результатів за допомогою команд

Нижче можемо побачити вкладку «Log», де виводиться запис про успішне чи не успішне кожної команди, а також час виконання, поруч з нею також є кнопка для видаленя повідомлень в журналі (рисунок 4.8).

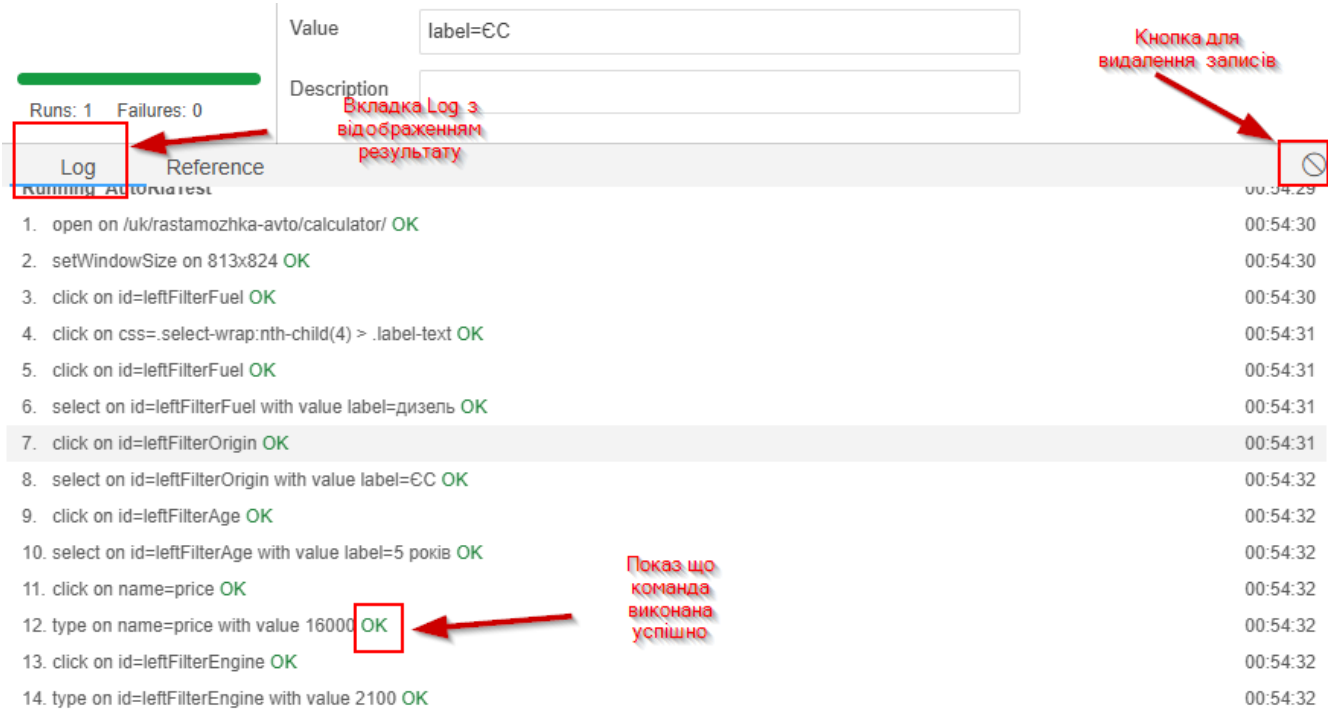

Рисунок 4.8 – Відображення результатів виконання команд

Також, мається вкладка «Reference» вона створена для того аби тестувальник міг отримати інформацію про виділену команду в таблиці команд зверху.

Після того як було введено всі необхідні параметри та натиснули кнопку «Розрахувати», отримуємо результат 45 732 EUR, необхідно перевірити фінальне значення – фактичний результат розрахунку та прорахунків окремих і ввести додатково ці значення. Щоб це перевірити на сторінці треба натиснути на суму розмитнення правою кнопкою миші, далі перейти до Selenium IDE та обрати параметр Verefity і далі text (рисунок 4.9). Після цього до списку команд додається: «`css=.item:nth-child(5) .casual`».

Це є стандартна перевірка, яка виконується при автоматизованих тестів і вона дає можливість перевірити, що калькулятор розмитнення виводить правильно значення.

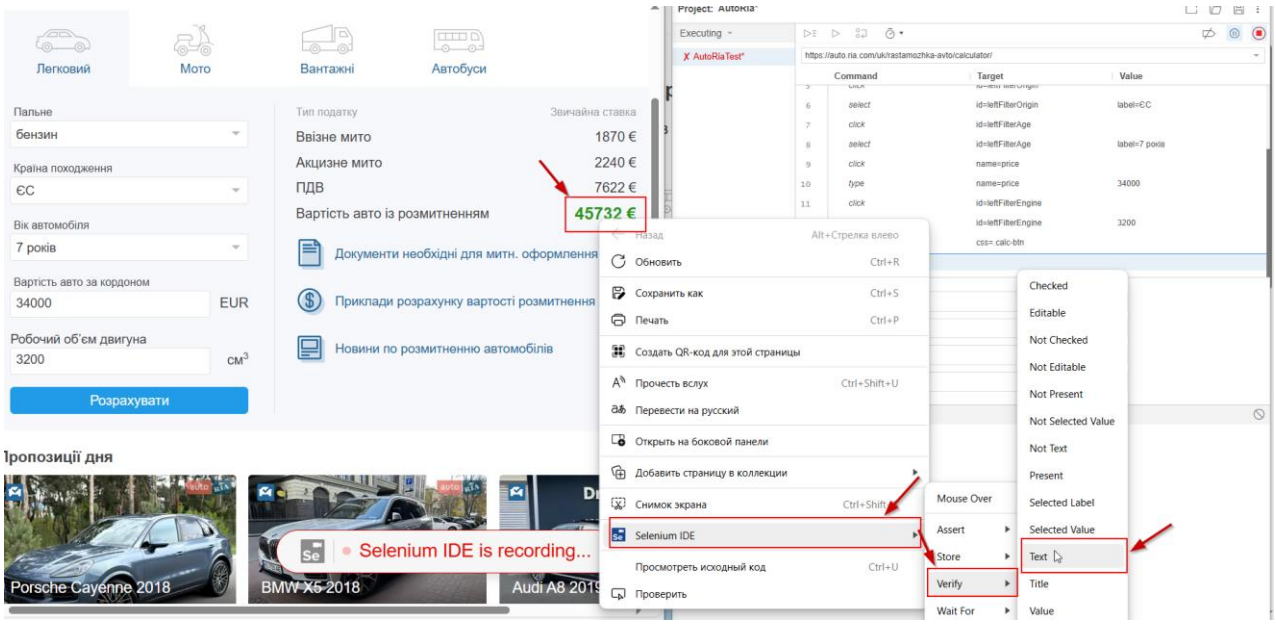

Рисунок 4.9 – Додавання команди verify text

Після виконання всіх кроків переходимо знову в Selenium IDE та виконуємо фінальний запуск тестів. Для цього натискаємо на кнопку запуску та очікуємо проходження всіх тестів, після успішного завершення тесту його результат має відображатись зеленим кольором та проставлені мітки «ОК» , саме за допомогою цього можна зрозуміти, що все вірно виконано (рисунок 4.10).

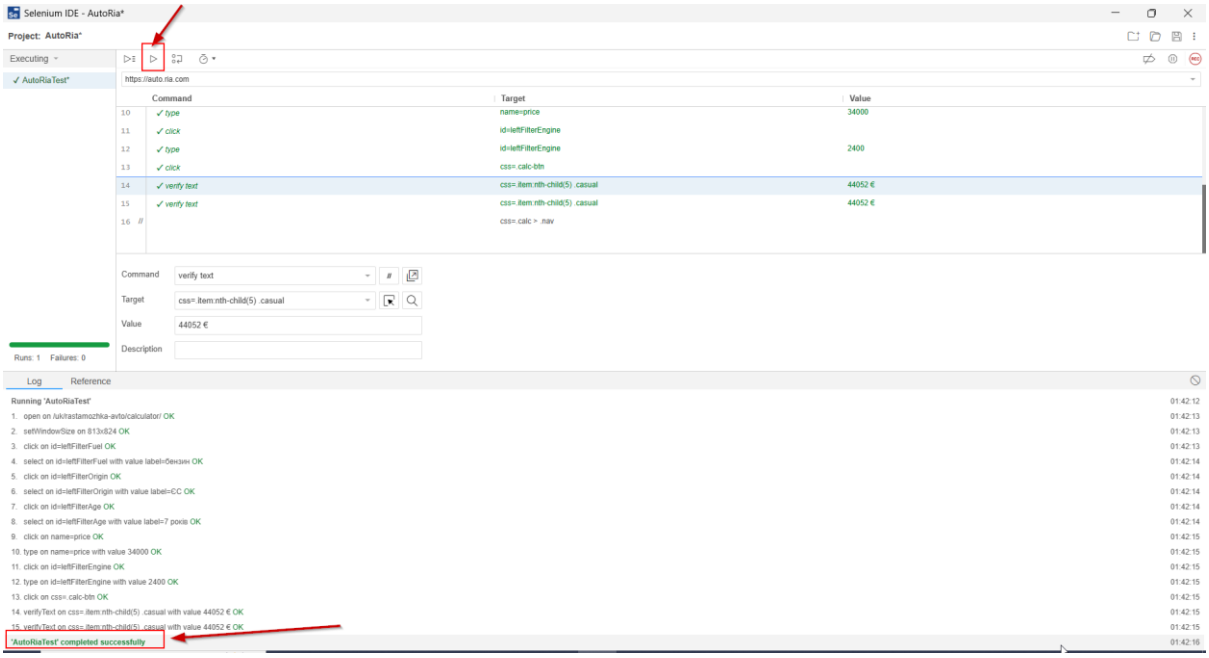

Рисунок 4.10 – Результат успішного виконання тесту.

Переходимо до наступного етапу та додаємо об’єднані тести, для яких раніше розробляли тестові сценарії. Першим в цьому списку було перевірка початкових значень, метою якого є переконатися, що при першому відкритті всі поля встановлені зі стандартним значенням, а там де текстові поля вони мають бути порожніми, для того аби не збивати користувача. Для цього необхідно створити спочатку новий тест та дати йому назву, наприклад InitTest (рисунок 4.11).

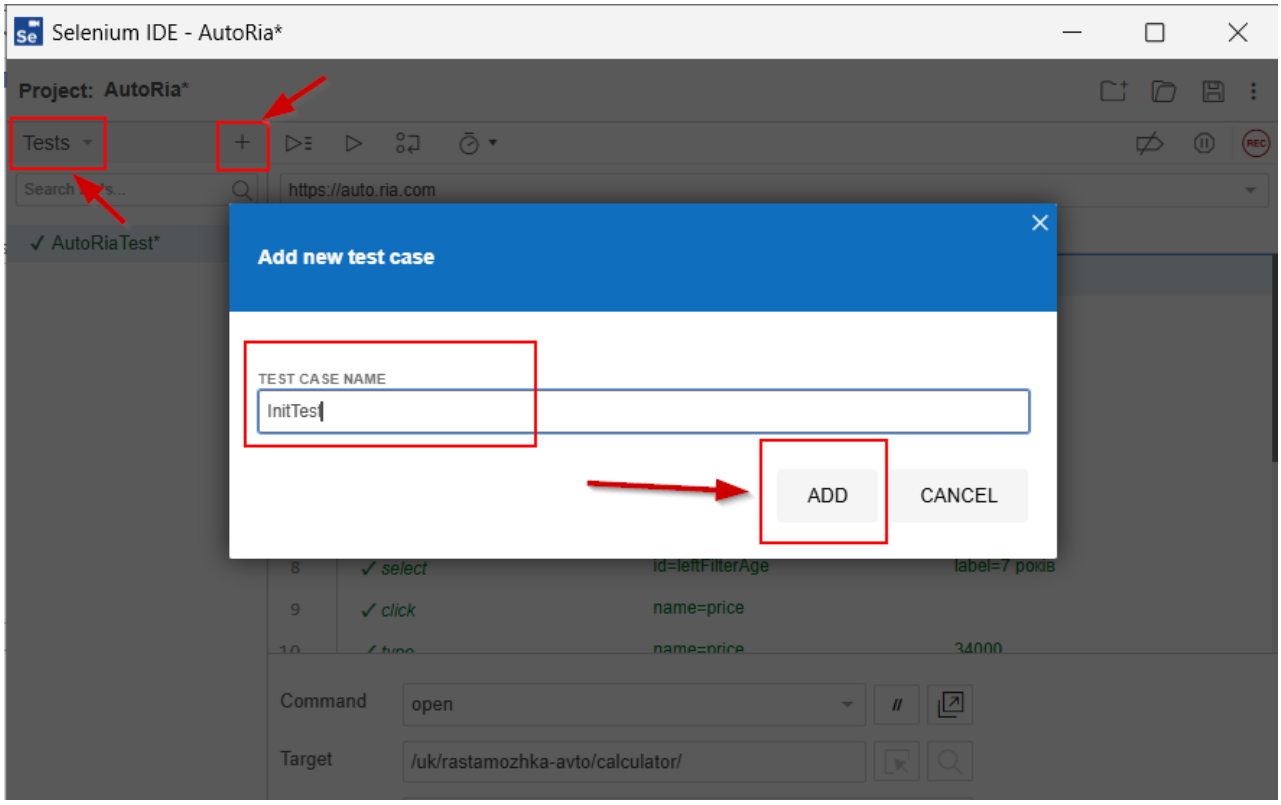

Рисунок 4.11 – Створення нового тесту для перевірки початкових значень

Після створення нового тесту для перевірки необхідно запустити режим запису тесту, перейти на сторінку та перевірити основні поля та тереконатися, що встановлені стандартні значення, а саме:

а) Пальне – бензин,

б) Країна походження – інше,

в) Вік автомобіля – 1 рік,

г) Вартість авто закордоном – пусте значення,

д) Робочий об’єм двигуна – пусте значення.

Після візуальної перевірки клікаємо правою кнопкою миші на поле для перевірки та обираємо Verifity і далі Value, зупиняємо записи та запускаємо тести, перевіряємо, що всі тести виконані успішні(підсвічено зеленим кольором) (рисунок 4.12).

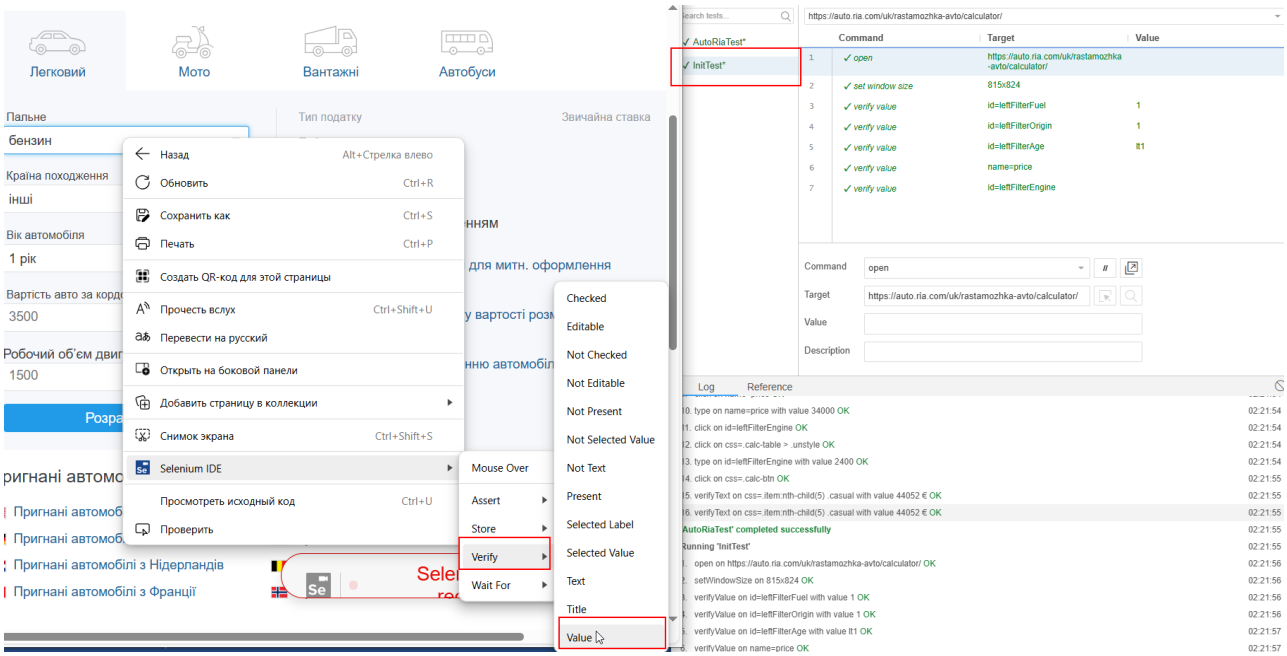

Рисунок 4.12 – Перевірка роботи InitTest

По аналогії з попереднім тестом необхідно створити новий тест, який буде перевіряти калькулятор на предмет невірно введених даних, назвемо його «NegativeTest», зможемо переконатися, що калькулятор буде коректно обробляти результат, якщо користувачем залишено пусті поля або ж введено некоректне значення у відповідне поле.

У поле з вартістю автомобіля додамо від’ємне значення для перевірки роботи з таким форматом, а в поле з об’ємом двигуна введемо цифри разом із символами. Очікується, що буде виведено помилку про невідповідність даних і користувача не проаустить до наступного кроку.

Після того як з’явиться помилка клікаємо на неї правою кнопкою миші та обираємо Selenium IDE , далі Verifity та Text (рисунок 4.13).

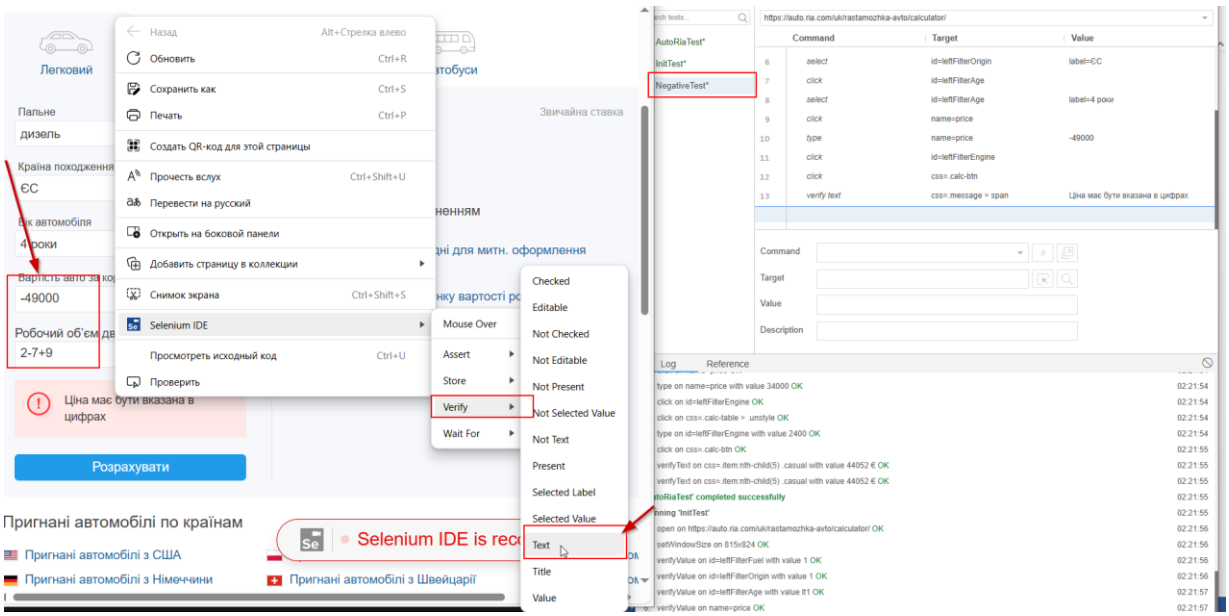

Рисунок 4.13 – Перевірка роботи NegativeTest

Створюємо ще один тест – NavigationTest, який буде перевіряти як працює перехід з однієї сторінки (початкової – калькулятор розмитнення) на інші сторінки, наприклад сторінку з оголошенням або ж « Приклад розрахунку вартості розмитнення». Перевірити відкриття одразу декількох сторінок, переконатися, що після повернення на сторінку де проводився розрахунок результат залишився збереженим. Виконати перевірку Verify Value (рисунок 4.14).

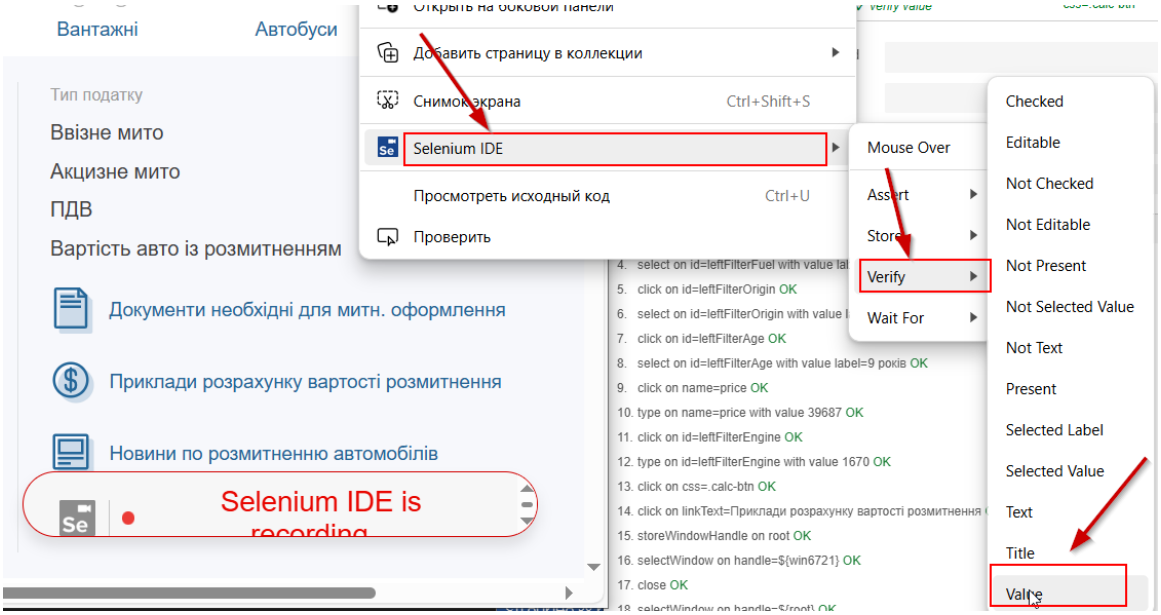

Рисунок 4.14 – Перевірка роботи NavigationTest

Створимо останній додатковий тест, який буде допомагати перевіряти, чи вірно відкривається сторінка, після того як користувач клікне на будь яке посилання чи кнопку зі сторінки калькулятора зозмитнення і чи буде вона мати потрібний та очікуваний заголовок <title> у HTML-документі Command: Verify Title .

Для цього перейдимо на сторінку калькулятора, пенатиснимо кнопку «Продати авто», після відкриття сторінки на будь-якому місці клікаю правою кнопкою миші та обираю Verify Title. Цей тест буде гарантувати, що посилання преведе користувача на правильну сторінку, а не порожню чи сторонню, також одразу перевіряє і маршрутизацію веб-застосунку та коректність завантаження сторінки (рисунок 4.15).

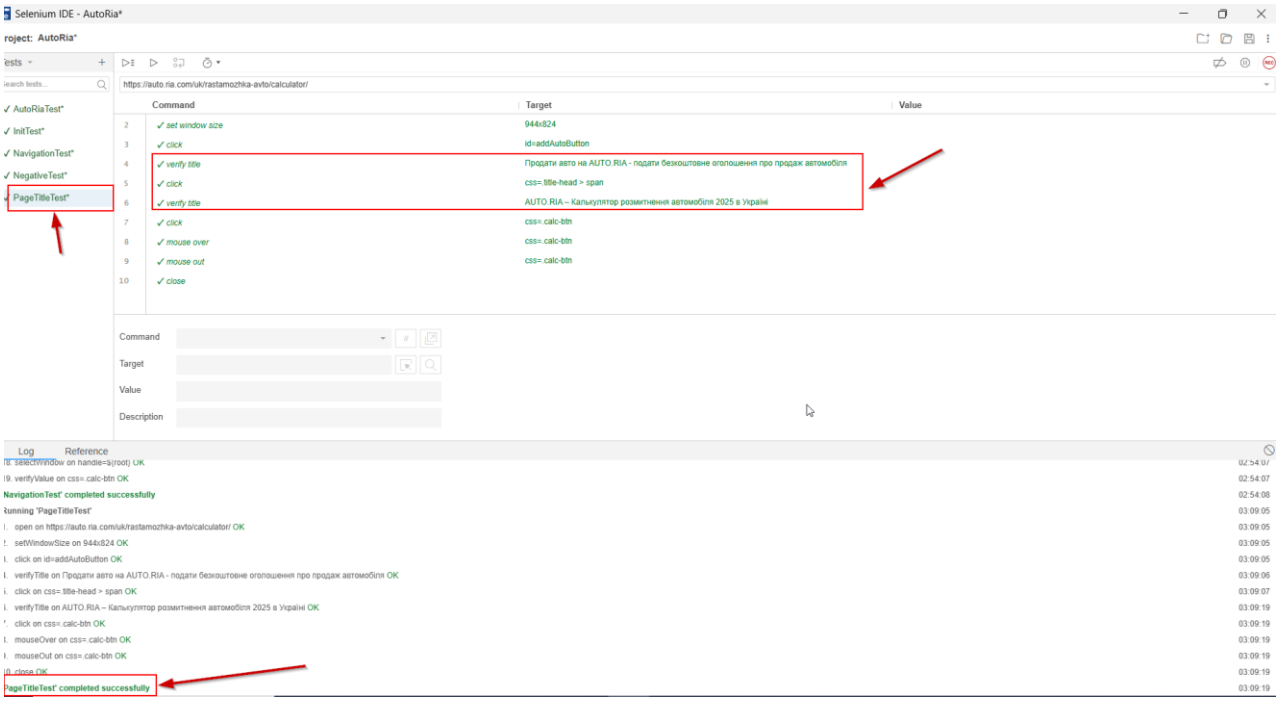

Рисунок 4.15 – Перевірка роботи PageTitleTest

Після того як маємо вже певну кількість тестів рекомендовано їх об’єднувати в тестовий набір, у Selenium IDE є така можливість, для цього необхідно переходити до Test Suite (рисунок 4.16).

Test Suite – це є група певних тестових сценаріїв, які об’єднуються у єдину структуру з метою побудови організованого і в той же час ефективного

автоматизованого тестування, це є дуже зручним, особливо, якщо буде занадто багато тестів, то їх сприйняття буде ускладненим і не потрібно буде запускати кожен тест вручну. Також, тестові набори дозаоляють запускати одразу всі тести за допомогою одного кліку та виконати всі перевірки автоматично, це значно заощаджує час при проведенні роботи. Як знаємо, дуже часто веб – застосунки потребують особливо тестування, а особливо перед запуском, тому саме за допомогою тестових наборів можна покращити цей проуес без участі людини. Коли тестувальник працює на великому проекті, то зазвичай тести групуються по сткутурі і їх можна буде об’єднувати за логікою, що допоможе знаходити швидко потрібні перевірки, навіть, якщо на проекті новий тестувальний, який не працював з проектом. Також, в такому форматі набагато легше переглянути звітність у журналі Log та відслідкувати, якщо сталась помилка під час виконання певного тесту.

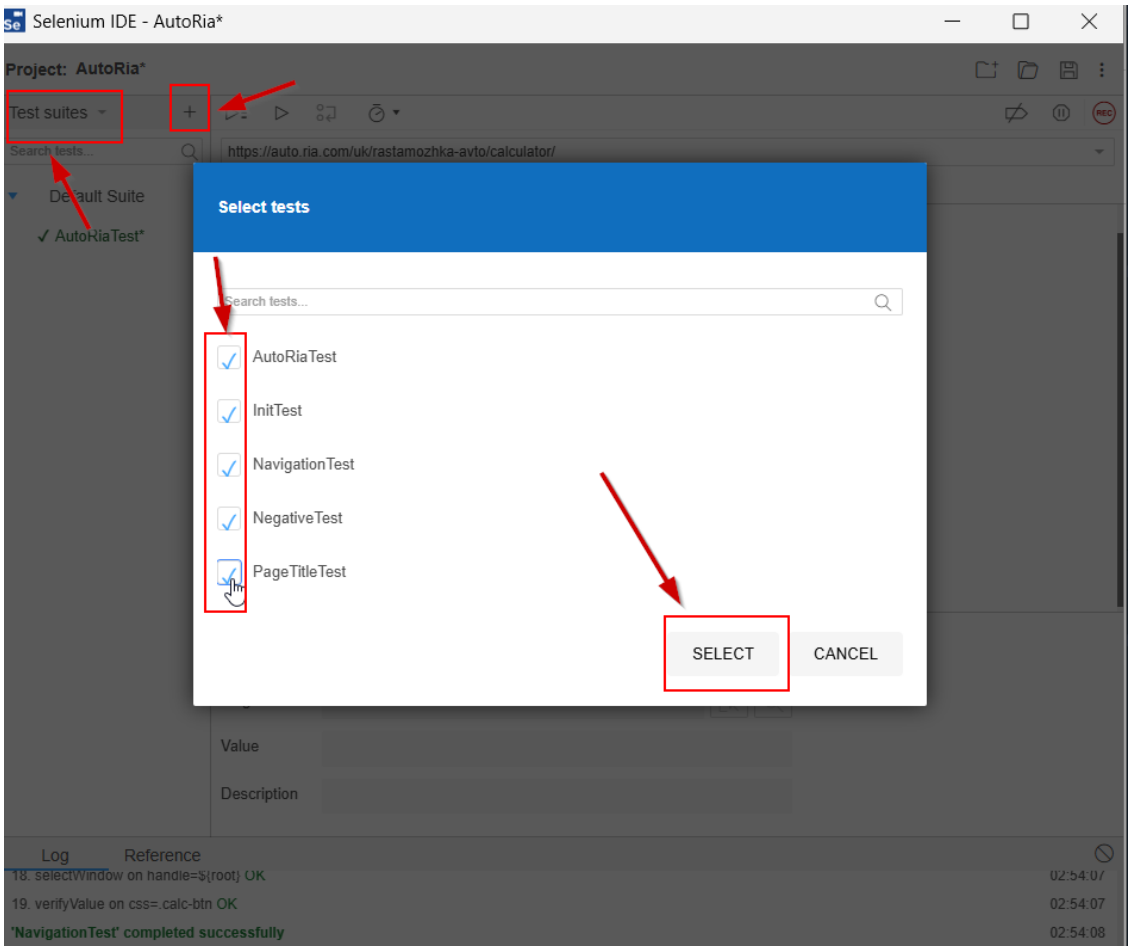

Рисунок 4.16 – Створення тестового набору

Після того як створили тестовий набір, рекомендовано перевірити, чи вірно тепер спрацьовує запуск. Тому, для цього в режимі тестових наборів натискаємо кнопку «Запуск тестів» та перевіряємо,щоб тести пройшли успішно.

4.4 Розробка автоматизованих тестів у Selenium WebDrive

Коли вже є наявні розроблені тести у Selenium IDE, то їх можна експортувати до Selenium WebDrive, при обраній мові програмування. Так як, Selenium IDE підходить для швидкого створення простих та гнучких тестів, де потрібна гнучкість, а саме експорт дозволяє використовувати ці тести на максимум. Але перед експортом необхідно переконатися, що тести працюють без помилок, оптимізовано локатори.

Загалом, є багато переваг до Selenium WebDrive:

а) Гнучкість: можливість використовувати функції, цикл, БД, умови та підключати API.

б) Покращене логування і наявність звітів.

в) Можливість маштабованості, наприклад, запуск на декількох браузарах, створення паралельного тестування.

Отже, для експортування необхідно відкрити Selenium IDE зі створеним тестовим набором, натиснути три крапки у меню та обрати пункт «Експорт», обрати необхідну мову, яку будемо використовувати (рисунок 4.17). Далі завантажити файл тесту, який містиме вже готовий скелет.

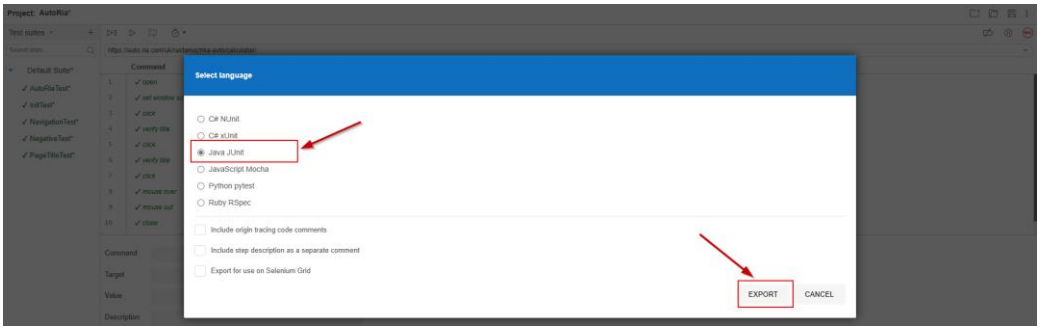

Рисунок 4.17 – Експорт тестів з вихідним кодом на Java JUnit

Після експортування файлу він зберігається до обраної папки і далі його необхідно відкрити за допомогою Intellij IDEA (рисунок 4.18).

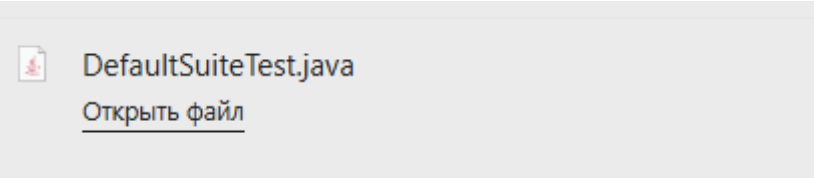

Рисунок 4.18 – Збережений файл з тестами з Selenium IDE

Після відкриття файлу він буде доданий до проекту, який створюємо та даємо йому назву згідно проекту «AutoRia». Це має бути проект з Maven, зберігаємо запропоновані налаштування з JDK. Після зберігається проект готовий до роботи, але перед цим необхідно додати залежності до файлу pom.xml (лістинг 4.1).

Лістинг 4.1 – Додавання залежностей до файлу pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>

 <groupId>com.autoria</groupId>
 <artifactId>autoria</artifactId>
 <version>1.0-SNAPSHOT</version>
 <packaging>jar</packaging>

 <name>autoria</name>
 <url>http://maven.apache.org</url>

 <properties>
  <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
 </properties>
<dependency>
 <groupId>org.junit.jupiter</groupId>
 <artifactId>junit-jupiter-engine</artifactId>
 <version>5.10.0</version>
 <scope>test</scope>
</dependency>
  <!--
https://mvnrepository.com/artifact/org.seleniumhq.selenium/selen
```

```
ium-java -->
  <dependency>
   <groupId>org.seleniumhq.selenium</groupId>
   <artifactId>selenium-java</artifactId>
   <version>4.30.0</version>
  </dependency>

  <!-- https://mvnrepository.com/artifact/org.testng/testng -->
  <dependency>
   <groupId>org.testng</groupId>
   <artifactId>testng</artifactId>
   <version>7.11.0</version>
   <scope>test</scope>
  </dependency>

  <dependency>
   <groupId>org.slf4j</groupId>
   <artifactId>slf4j-api</artifactId>
   <version>2.0.9</version>
  </dependency>
  <dependency>
   <groupId>org.slf4j</groupId>
   <artifactId>slf4j-simple</artifactId>
   <version>1.7.36</version>
   <scope>test</scope>
  </dependency>
 </dependencies>

 <build>
  <plugins>
   <!-- Запуск тестів через mvn clean install/test -->
   <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>3.0.0-M5</version>
   </plugin>
  </plugins>
 </build>
</project>
```

Залежності, які прописано вище було додано до проєкту бібліотеку з Selenium на мові програмування Java, також, за допомогою неї можна прауювати та керувати браузерами при виконанні автоматизованого тестування. Наступною доданою залежністю є slf4j-api – як можливість для логування і забезпечує універсальний інтерфейс, який використовується для різноманітних систем журналювання. Також , підключається залежність testng, яка підключається для організції автоматизованого тестування, та дає

можливість створення групування, параметризовувати, робити залежності між тестами.

Далі, необхідно створити пакет Java з назвою відповідною до тестування, наприклад – autoria.calculator, до цього файлу переносимо тести, які були експортовані з Selenium IDE. Як відомо, Selenium IDE за стандартними налаштуваннями експортує тести з проекту з використанням попередньої версії JUnit, тому її необхідно вручну оновити та імпортувати JUnit5 (лістинг 4.2).

Лістинг 4.2 – Імпортування JUnit5

```
package com.autoria.calculator;

import org.junit.jupiter.api.*;
import org.openqa.selenium.*;
import org.openqa.selenium.chrome.ChromeDriver;
import io.github.bonigarcia.wdm.WebDriverManager;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;

import java.time.Duration;

import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.Matchers.is;

@TestInstance(TestInstance.Lifecycle.PER_CLASS)
public class AutoRiaTest {

  private WebDriver driver;
  private WebDriverWait wait;

  @BeforeAll
  void setupDriver() {
    WebDriverManager.chromedriver().setup(); // автоматично
завантажує відповідний chromedriver
  }

  @BeforeEach
  void init() {
    driver = new ChromeDriver();
    wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    driver.manage().window().setSize(new Dimension(813, 824));
  }

  @AfterEach
  void tearDown() {
```

```
    if (driver != null) {
      driver.quit();
    }
  }

  @Test
  void autoRiaTest() {
    driver.get("https://auto.ria.com/uk/rastamozhka-
avto/calculator/");

    WebElement fuel =
driver.findElement(By.id("leftFilterFuel"));
    fuel.click();
    fuel.findElement(By.xpath("//option[. =
'бензин']")).click();

    WebElement origin =
driver.findElement(By.id("leftFilterOrigin"));
    origin.click();
    origin.findElement(By.xpath("//option[. = 'ЄС']")).click();

    WebElement age = driver.findElement(By.id("leftFilterAge"));
    age.click();
    age.findElement(By.xpath("//option[. = '7
років']")).click();

    WebElement price = driver.findElement(By.name("price"));
    price.click();
    price.sendKeys("34000");

    WebElement engine =
driver.findElement(By.id("leftFilterEngine"));
    engine.click();
    engine.sendKeys("2400");

    driver.findElement(By.cssSelector(".calc-btn")).click();

    String result = wait.until(ExpectedConditions
      .visibilityOfElementLocated(By.cssSelector(".item:nth-
child(5) .casual")))
      .getText();

    assertThat(result, is("44052 €"));
  }
}
```

За допомогою цього тесту було реалізовано автоматичний тест для тестування веб-застосунку, а саме калькулятора розмитнення авто на сайті AutoRia, в тесті використовується JUnit 5 і Selenium WebDriver. Він запускається реальним браузером Chrome через WebDriverManager. Тест

відтворює роботу реального користувача, який переходить на сторінку та виконує взаємодію з елементами заповнення форми. В ньому використовуються різні локатори для основних полів у формі, взаємодії з випадаюсими списками та отримання кнопки результатами, все це робиться за допомогою: By.id(), By.xpath(), By.cssSelector(). За допомогою тесту виконується перевірка по розрахунку вартості і він повністю відповідає очікуваному результату. Отже, тест проходить успішно і відповідає розрахованому значенню.

Перейдемо до наступного тесту та перетворень для нього, тест буде виглядати наступним чином (лістинг 4.3).

Лістинг 4.3 – Тест 2 з використанням JUnit5

```
package com.autoria.calculator;

import io.github.bonigarcia.wdm.WebDriverManager;
import org.junit.jupiter.api.*;
import org.openqa.selenium.*;
import org.openqa.selenium.chrome.ChromeDriver;

import java.util.HashMap;
import java.util.Map;

import static org.junit.jupiter.api.Assertions.*;

@TestInstance(TestInstance.Lifecycle.PER_CLASS)
public class InitTest {

  private WebDriver driver;
  private Map<String, Object> vars;
  private JavascriptExecutor js;

  @BeforeAll
  public void setupClass() {
    WebDriverManager.chromedriver().setup();
  }
  @BeforeEach
  public void setUp() {
    driver = new ChromeDriver();
    js = (JavascriptExecutor) driver;
    vars = new HashMap<>();
    driver.manage().window().setSize(new Dimension(816, 824));
  }
  @AfterEach
  public void tearDown() {
```

```
    if (driver != null) {
      driver.quit();
    }
  }
  @Test
  public void initTest() {
    driver.get("https://auto.ria.com/uk/rastamozhka-
avto/calculator/");

    // Введення некоректної вартості авто
    WebElement priceField =
driver.findElement(By.name("price"));
    priceField.click();
    priceField.sendKeys("-49000");

    WebElement engineField =
driver.findElement(By.id("leftFilterEngine"));
    engineField.click();

    driver.findElement(By.cssSelector(".calc-btn")).click();

    String enteredPrice = priceField.getAttribute("value");
    vars.put("enteredPrice", enteredPrice);
    assertNotEquals("-49000", enteredPrice, " Ціна має бути
вказана в цифрах ");
  }
}
```

Тест перевіряє надійність та коректність роботи застосунку, також перевіряє реакцію калькулятора, якщо користувач випадково введе некоректні дані, наприклад як в тесті від'ємну вартість. Після вдосконалення коду який було надано першочергово і додано WebDriverManager і JUnit 5, які дозволяють в подальшому вдосконалювати тести, підтримувати та розширювати їх, він обробляє явно неможливі значення які можуть привести до помилок в подальшому (лістинг 4.4).

Лістинг 4.4 - Тест 3 з JUnit5

```
import static org.junit.jupiter.api.Assertions.*;

public class NavigationTest {
  private WebDriver driver;
  private Map<String, Object> vars;
  private JavascriptExecutor js;
  @BeforeAll
  public static void setupClass() {
    WebDriverManager.chromedriver().setup();
  }
```

```
  @BeforeEach
  public void setUp() {
    driver = new ChromeDriver();

driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10)
);
    driver.manage().window().maximize();
    js = (JavascriptExecutor) driver;
    vars = new HashMap<>();
  }
  @AfterEach
  public void tearDown() {
    driver.quit();
  }
  @Test
  public void navigationTest() {
    driver.get("https://auto.ria.com/uk/rastamozhka-
avto/calculator/");

    WebElement fuelDropdown =
driver.findElement(By.id("leftFilterFuel"));
    fuelDropdown.click();
    fuelDropdown.findElement(By.xpath("//option[. =
'дизель']")).click();

    WebElement originDropdown =
driver.findElement(By.id("leftFilterOrigin"));
    originDropdown.click();
    originDropdown.findElement(By.xpath("//option[. =
'ЄС']")).click();

    WebElement ageDropdown =
driver.findElement(By.id("leftFilterAge"));
    ageDropdown.click();
    ageDropdown.findElement(By.xpath("//option[. = '9
років']")).click();
    WebElement priceInput =
driver.findElement(By.name("price"));
    priceInput.clear();
    priceInput.sendKeys("39687");
    WebElement engineInput =
driver.findElement(By.id("leftFilterEngine"));
    engineInput.clear();
    engineInput.sendKeys("1670");
    WebElement calcButton =
driver.findElement(By.cssSelector(".calc-btn"));
    calcButton.click();
    vars.put("window_handles", driver.getWindowHandles());

    // Переходимо за посиланням на приклади розрахунку
    driver.findElement(By.linkText("Приклади розрахунку вартості
розмитнення")).click();
    String newWindowHandle = waitForWindow(2000);
```

```
    vars.put("new_window", newWindowHandle);
    driver.switchTo().window(vars.get("new_window").toString());
    driver.close();
driver.switchTo().window(driver.getWindowHandles().iterator().ne
xt());

    // Перевіряємо, що кнопка "Розрахувати" все ще має
правильний текст
    String buttonValue =
driver.findElement(By.cssSelector(".calc-
btn")).getAttribute("value");
    assertEquals("Розрахувати", buttonValue);

  }
}
```

За допомогою цього тесту відбувається перехід між новими вкладками, закриття її, при цьому спрацьовує перевірка, щоб розрахунки зберігалися вісля повернення на сторінку. За допомогою частини з JUnit 5 та анотаціям від нього використовується WebDriverManager, який автоматично підтягує необхідний драйвер під обрану версію. Також, створюється новий екземпляр за допомогою «@BeforeEach» перед кожним тестом, а після завершення кожного тесту виконужться анотація «@AfterEach», яка закриває браузер. У результаті буде введено послідовно значення у всі наявні поля в калькуляторі, тест перевіряє коректність роботи основного вікна після додаткового переходу та закриття нової відкритої сторінки. Якщо є навантаження суттєве на сервері, то може бути затримка результату, тому використовується явне очікування для перевірки вірності результату.

Останнім реалізованим тестом буде тест за допомогою якого відбувається перевірка на коректність заголовків на сторінці при переході між сторінкою на створення оголошення та калькулятором розмитнення (лістинг 4.5).

Лістинг 4.5 – Тест 4 з використанням JUnit5

```
public class CustomsCalcPageTitleTest {
  private WebDriver driver;
  private WebDriverWait wait;
  @BeforeAll
  public static void setupClass() {
```

```
    WebDriverManager.chromedriver().setup();
  }
  @BeforeEach
  public void setUp() {
    driver = new ChromeDriver();
    driver.manage().window().setSize(new Dimension(944, 824));
    wait = new WebDriverWait(driver, Duration.ofSeconds(10));
  }

  @AfterEach
  public void tearDown() {
    if (driver != null) {
      driver.quit();
    }
  }
  @Test
  public void pageTitleTest() {
    // Відкриття сторінки калькулятора розмитнення
    driver.get("https://auto.ria.com/uk/rastamozhka-
avto/calculator/");

        WebElement sellButton =
wait.until(ExpectedConditions.elementToBeClickable(By.id("addAut
oButton")));
    sellButton.click();

    // Перевірка заголовка сторінки після переходу
    String sellPageTitle = driver.getTitle();
    assertEquals("Продати авто на AUTO.RIA - подати безкоштовне
оголошення про продаж автомобіля", sellPageTitle);
    WebElement backButton =
wait.until(ExpectedConditions.elementToBeClickable(By.cssSelecto
r(".title-head > span")));
    backButton.click();
       String calcPageTitle = driver.getTitle();
    assertEquals("AUTO.RIA – Калькулятор розмитнення автомобіля
2025 в Україні", calcPageTitle);
    WebElement calcButton =
wait.until(ExpectedConditions.elementToBeClickable(By.cssSelecto
r(".calc-btn")));
    calcButton.click();
    Actions actions = new Actions(driver);
    actions.moveToElement(calcButton).perform();
actions.moveToElement(driver.findElement(By.tagName("body")), 0,
0).perform();
  }
}
```

В тесті використовується явне очікування, для того аби попередити помилки, які можливі під час виконання тесту, якщо раптом буде повільне завантаження елементів. Також, тест перевіряє правильність переходів між обраними сторінками, ну і звичайно ж виконує основну перевірку для якої

він був створений, а саме заголовки сторінок та взаємодію між елементами.

Після того як було відредаговано всі тести необхідно їх об’єднати до тестового набору, як робили це попередньо в Selenium IDE, для того аби було зручно проводити тестування, тому, об’єдную їх до одного набору AutoRiaCalculatorTestSuite (лістинг 4.6).

Лістинг 4.6 – Об’єднання тестів до тестового набору

```
package ua.edu.znu.autoriacalculator;
import org.junit.platform.suite.api.SelectClasses;
import org.junit.platform.suite.api.Suite;
@Suite
@SelectClasses({AutoRiaTest.class,PageTitleTest.class,NegativeTe
st.class,NavigationTest.class,InitTest.class})
public class AutoRiaCalculatorTestSuite {
}
```

Після цього виконується запуск тестового набору, який перевірить повністю роботу веб-застосунку « Калькулятор розмитнення автомобіля» відповідно до прописаних в тестах перевірок. В результаті тести пройшли успішно, тому всі перевірки були виконані коректно.

4.5 Аналіз результатів тестування

Після того як було проведено тестування функцій калькулятора розмитнення на AutoRia спочатку з використанням Selenium IDE і далі за допомогою Selenium WebDriver та JUnit 5, можна сказати, що тза допомогою наявних тестів було перевірено основний функціонал роботи калькулятора, було визначено, що система працює коректно та відповідає вимогам, які будуть дозволяти комфортно взаємодіяти користувачу з основними функціями та покривати наявні потреби в розрахунку. Хоча і з технічного боку система працює коректно рекомендується покращити логічні підказки для користувачів при введенні невірних даних, тобто більш точно підсвічувати в якому полі допущено помилку та виводити актуальний текст.

В подальшому можна покращити тестування та розширити тести, для

того аби покрити більший функціонал, що дасть змогу перевірити більшу кількість типових сценаріїв, які може задати користувач в ході роботи. Саме цей варіант допоможе максимально виявити потенційно вразливі місця системи, а також покращити стабільність роботи. Рекомендується також оптимізувати загальний час виконання, в чому може допомогти використання headless-режим. А також, необхідно додати перевірки для крос-браузерної сумісності для виявлення специфічних помилок в роботі.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було опрацьовано необхідну літературу для досягання основної мети роботи і в результаті розроблено та проведено автоматизовані тести на прикладі калькулятора розмитнення автомобілів на платформі AutoRia, за допомогою технологій Selenium WebDriver та JUnit. Детально вивчено фреймворк Selenium та його основні можливості, що дало можливість в подальшому створювати автоматизовані тести.

Також, було реалізовано повний тестовий набір для тестування, а саме: функіональні тести, навігаційні, негативні тести та інші. Створено подальший план для покращення наявних тестів, а також для розробки нових сценаріїв та наборів тестів, які допоможуть удосконалити систему калькулятора та зробити його ще більш інтуїтивно зрозумілим для користувача.

В ході практичної реалізації тестів було показано високу ефективність автоматизованого тестування за допомогою використання фреймворку Selenium для перевірки веб-застосунків. Такі тести можуть проводитись без прямої участі людини, що значно скоротить витрачений час та спростить процес тестування.

Отже, виконана кваліфікаційна робота ще раз підтвердила важливість та доцільність в сучасному автоматизованому тестуванні технології Selenium. Отримані результати підтверджують, що тестування є важливим та необхідним етапом у життєвому циклі розробки та дозволяє значно підвищити кінцеву якість продукту і зменшити ризики виникнення критичних помилок. У подальшому, планується розширеня тестового покриття, що дозволить використовувати тести не лише для окремої сторінки калькулятора, а й в цілому системи, враховуючи можливі оновлення платформи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Офіційна документація по Selenium. URL: https://www.selenium.dev/documentation/ (дата звернення: 27.05.2025).
2. García Boni. Selenium WebDriver 4 Practical Guide. Packt, 2022. 137 с.
3. Richardson A. Java WebDriver: Test Automation and Development.78 с.
4. Офіційний вебсайт: Міністерство фінансів України. Державна митна служба. Митне оформлення автомобілів. URL: https://customs.gov.ua/mitne-oformlennia-avtomobilia (дата звернення: 01.06.2025).
5. Курс «Створення фреймворка і автоматизація тестів на Java+Selenium». URL: https://coursehunter.net/category/selenium (дата звернення: 28.06.2025).
6. Stack Overflow. Selenium WebDriver. URL: https://stackoverflow.com/questions/tagged/selenium-webdriver (дата звернення: 05.06.2025).
7. Burns S. Selenium Design Patterns and Best Practices, 2015. 350 с.