КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Кваліфікаційна робота

на здобуття освітнього рівня бакалавра
за спеціальністю 121 Інженерія програмного забезпечення
на тему:

АВТОМАТИЗАЦІЯ ТЕСТУВАННЯ ВЕБ-ЗАСТОСУНКІВ

Виконала студентка 4-го курсу
Марія КУШНІРЕНКО _____
(підпис)

Науковий керівник:
доцент, кандидат фізико-математичних наук
Оксана ШКІЛЬНЯК _____
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри інтелектуальних програмних систем
«___»_____202_р.,
протокол №____
Завідувач кафедри
Олександр ПРОВОТАР _____
(підпис)

Київ – 2021

РЕФЕРАТ

Обсяг роботи 58 сторінок, 21 ілюстрація, 17 джерел посилань.

АВТОМАТИЗАЦІЯ ТЕСТУВАННЯ, ВЕБ-БРАУЗЕР, ВЕБ-ДРАЙВЕР, ВЕБ-ЕЛЕМЕНТ, ВЕБ-ЗАСТОСУНОК, ЛОКАТОР, ТЕСТ-КЕЙС, ФРЕЙМВОРК

Об’єктом роботи є дослідження та аналіз підходів до створення систем, що призначені для автоматизованого тестування веб-застосунків та інструментарію для цього. Предметом роботи є набір автоматизованих тест-кейсів для тестування інтернет-магазину з використанням обраного інструментарію.

Метою роботи є створення власного набору автоматизованих тест-кейсів для тестування інтернет-магазину з використанням найкращих підходів для їх написання.

Методи розроблення: імітація дій користувача за допомогою спеціального тестового фреймворку. Інструменти розроблення: комерційне інтегроване середовище розробки IntelliJ IDEA Ultimate Edition (безкоштовно поширюване для студентів), мова програмування Java, інструмент для автоматизації дій веб-браузера Selenium WebDriver.

Результати роботи: проаналізовано переваги і недоліки найпопулярніших інструментів автоматизації дій веб-браузера та шаблонів написання автоматизованих тест-кейсів, розроблений автоматизований набір тест-кейсів для тестування основної функціональності інтернет-магазину «ROZETKA».

Даний автоматизований набір тест-кейсів, може використовуватися як приклад написання GUI тестів в рамках курсу «Програмна інженерія» та в майбутньому навчальної дисципліни «Тестування програмного забезпечення».

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

GUI (Graphical User Interface) − графічний інтерфейс користувача

API (Application Programming Interface) − прикладний програмний інтерфейс

Десктоп − настільний комп'ютер

ОС − операційна система

DOM (Document Object Model) − об'єктна модель документа

DevTools (Developer Tools) − інструменти розробника

HTML (HyperText Markup Language) − мова розмітки гіпертексту

CSS (Cascading Style Sheets) − каскадні таблиці стилів

URL (Uniform Resource Locator) − єдиний вказівник на ресурс

ООП − об'єктно орієнтоване програмування

SOLID − абревіатура складена з перших літер п'яти базових принципів об'єктно-орієнтованого програмування та дизайну

MVC (Model-view-controller) − модель-представлення-контролер

TDD (Test-driven development) − керована тестами розробка

JVM (Java Virtual Machine) − віртуальна машина Java

ВСТУП

Оцінка сучасного стану об’єкта розробки. Сучасні гнучкі методології програмування призвели до активного розвитку засобів автоматизованого тестування. Характерною рисою таких методологій є отримання великої кількості різних версій продукту (після кожної ітерації), які потрібно протестувати. Зазвичай це регресійне тестування, яке буде перевіряти працездатність вже перевіреної, працюючої на попередніх ітераціях, функціональності, а це великий шматок однотипної, повторюваної роботи, яку зручно делегувати із людини на машину. Крім цього, для кожної нової версії додаються нові тестові випадки, а також можуть змінюватися вже існуючі, тому потрібно розробляти такий засіб автоматизації тестування, що дозволятиме максимально швидко та з мінімальними витратами доповнити існуючий набір автоматизованих тест-кейсів. Саме для цього використовуються напрацьовані роками найкращі практики проєктування систем автоматизованого тестування, що називаються шаблонами.

Актуальність роботи та підстави для її виконання. Активний розвиток та модернізація засобів автоматизації тестування робить актуальним завдання пошуку найоптимальніших методів проєктування систем автоматизованого тестування. Для цього потрібна інформаційна технологія, яка може спростити процес створення та оновлення автоматичних тестів та сценаріїв. Саме тому доцільно проводити дослідження доступного на даний момент інструментарію та підходів до проєктування автоматизованого набору тест-кейсів і на базі цього дослідження створити власну систему для автоматизованого тестування веб-застосунку.

Мета й завдання роботи. Метою дипломної роботи є дослідження інструментів автоматизації тестування, а також актуальних найкращих практик

для створення таких систем, результатом якого має стати створення власного набору автоматизованих тест-кейсів. Для досягнення цієї мети було поставлено наступні завдання:

– дослідити області застосування автоматизації тестування;

– дослідити технології автоматизації тестування;

– дослідити інструменти автоматизації тестування;

– дослідити шаблони автоматизації тестування;

– розробити структуру системи автоматизованого тестування інтернет-магазину;

– розробити набір автоматизованих тест-кейсів для інтернет-магазину «ROZETKA».

Об'єкт, методи й засоби розроблення. Об’єктом розроблення системи автоматизованого тестування інтернет-магазину є процес тестування основної функціональності веб-сайту «ROZETKA».

Розробці даної системи передував аналіз доступних на даний момент інструментів для автоматизації дій веб-браузера та шаблонів проєктування автоматизованих наборів тест-кейсів. Основою для цього стало дослідження засобів автоматизації тестування та найкращих практик проєктування автоматизованих тестів.

Під час розробки системи використана еволюційна модель, заснована на наступних принципах: розробляється початковий мінімальний набір тест-кейсів, потім в ньому відбуваються структурні зміни для покращення розуміння і спрощення підтримки наростаючого набору тест-кейсів.

Інструментами створення системи було обрано комерційне інтегроване середовище розробки IntelliJ IDEA Ultimate Edition (безкоштовно поширюване для студентів), мова програмування Java та інструмент для автоматизації дій веб-браузера Selenium WebDriver.

Можливі сфери застосування. Розроблена система може застосовуватися як приклад автоматизованого набору GUI тест-кейсів у навчальному процесі в

рамках курсу «Програмна інженерія», під час вивчення теми тестування програмного забезпечення, та в майбутньому при опануванні навчальної дисципліни «Тестування програмного забезпечення».

Взаємозв'язок з іншими роботами. За методами розробки та інструментальними засобами робота виконувалася сумісно з роботами із наступних дисциплін: «Розробка WEB-орієнтованих систем» та «Програмна інженерія».

РОЗДІЛ 1. ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО АВТОМАТИЗАЦІЮ ТЕСТУВАННЯ. ІСТОРІЯ ВИНИКНЕННЯ

1.1 Тестування та його історія

Тестування програмного забезпечення [1] – це процес аналізу програмного засобу та супутньої документації з метою виявлення дефектів і підвищення якості продукту.

Тестування програмного забезпечення зароджувалося ще у середині минулого сторіччя і зазнало значних змін за час свого існування. Саме тому варто розглянути його еволюцію поетапно.

У 50-60-х роках тестування програмного забезпечення мало математичну основу та полягало у налагодженні програм. У цей період була актуальною концепція «вичерпного тестування», яка полягала у перевірці всіх можливих шляхів виконання програмного коду при всіх можливих вхідних даних. Але згодом дана концепція стала незастосовною, оскільки нереально осягнути всі варіанти вхідних даних, бо їх може бути нескінченна кількість.

У 70-х роках зародилися дві фундаментальні ідеї тестування: позитивне тестування на негативне тестування. Позитивне тестування дозволяє переконатися, що програма відповідає вказаним вимогам. Негативне тестування дозволяє визначити умови, за яких програма веде себе некоректно. Хоча дані ідеї є доволі суперечливими, але багато спеціалістів з галузі тестування справедливо зазначають, що ці ідеї насправді є взаємодоповнюючими цілями тестування.

У 80-х роках тестування поширилось із кінцевої стадії розробки програмного забезпечення на інші етапи. Це значно покращило якість кінцевого продукту, оскільки деяких проблем вдалося уникнути шляхом їхнього передбачення і вживання відповідних заходів для їх запобігання. Інші ж

проблеми стали виявлятися на більш ранніх стадіях розробки програмного забезпечення та їх стало легше і дешевше усувати. У цей період відбувся стрімкий розвиток та формалізація методологій тестування, а також почала зароджуватися автоматизація тестування.

У 90-х роках змістився акцент із тестування програмного забезпечення на більш широкий процес «забезпечення якості». Даний процес охоплює весь цикл розробки програмного забезпечення, що включає в себе наступні стадії: планування, проєктування, створення та виконання тест-кейсів та підтримку існуючих тестових середовищ. На цьому етапі тестування перейшло на якісно новий рівень розвитку, який, у свою чергу, спонукав подальший активний розвиток методологій тестування. З'явилися потужні інструменти управління процесом та автоматизацією тестування. Ці інструменти ще 30 років тому мали вигляд подібний до сучасних аналогів.

На початку двохтисячних з'явились гнучкі методології розробки програмного забезпечення та «керована тестами розробка». Автоматизація тестування стала невід'ємною частиною більшості проєктів. На цьому етапі змінюється ціль тестування: якщо раніше вона полягала у тому, що треба було перевірити відповідність програми вимогам, то зараз акцент змістився на те, що система має надавати кінцевому користувачу можливість ефективно вирішувати поставлені перед ним задачі.

Сучасний етап розвитку тестування характеризується наступними поняттями: гнучкі методології, гнучке тестування, глибока інтеграція із процесом розробки, широке використання автоматизації, колосальний набір технологій та інструментів та крос-функціональність команди.

1.2 Автоматизація тестування

Автоматизація тестування [1] − це набір технік, підходів та інструментальних засобів, що дозволяють виключити людину із виконання деяких задач у процесі тестування.

1.2.1 Переваги автоматизації

Швидкість виконання тест-кейсів може в рази або навіть на порядки перевищувати можливості людини. Наприклад, для того, щоб людині перевірити декілька файлів розміром в декілька десятків мегабайт кожен, то їй знадобиться на це декілька місяців. Для виконання цього ж завдання з використанням автоматизації людині знадобиться лише декілька секунд для того, щоб запустити відповідний код, і він виконається за декілька хвилин, а то і швидше.

Відсутність людського фактору (втома, неуважність, і тому подібне) під час виконання тест-кейсів також є перевагою автоматизованого тестування. Наприклад, яка ймовірність того, що людина допустить помилку, порівнюючи посимвольно два тексти розміром по 200 сторінок кожен, а таких текстів може бути не один. Перевірки потрібно виконувати одну за одною… Тому можна стверджувати, що людина гарантовано десь зробить помилку, а комп’ютер не помиляється.

Засоби автоматизації можуть виконати тест-кейси, які є надто складними, об’ємними чи швидкими для людини. Людина не може дозволити собі витрачати місяці або навіть роки, виконуючи вкрай складні рутинні завдання, під час яких гарантовано будуть допущені помилки.

Іншим яскравим прикладом тестування, яке не може бути проведено вручну, є тестування продуктивності. Під час нього необхідно з високою швидкістю виконувати певні дії, а також одночасно фіксувати значення широкого набору параметрів.

За допомогою автоматизації можна збирати, зберігати, аналізувати та представляти у зручній для людини формі дуже великі об’єми даних. Наприклад, журнали роботи систем автоматизованого тестування можуть мати розмір десятків гігабайт по кожній ітерації, а розумно налаштоване середовище автоматизації може самостійно проаналізувати такі об’єми даних, і користувач отримає лаконічні звіти на пару сторінок. Вони будуть містити зручні для розуміння даних графіки та таблиці, а також можливість занурюватися в деталі по мірі необхідності.

Засоби автоматизації здатні виконувати низькорівневі дії з додатком, операційною системою, каналами передачі даних і тому подібним. За допомогою автоматизації можна впливати на середовище виконання додатка чи навіть сам додаток, емулюючи типові події (наприклад, брак оперативної пам’яті чи процесорного часу) та фіксувати реакцію додатка.

Тобто, завдяки автоматизації ми отримуємо можливість збільшити тестове покриття за рахунок:

- виконання тест-кейсів, які неможливо пройти без автоматизації;
- багаторазового повторення тест-кейсів із різними вхідними даними;
- звільнення часу на створення нових тест-кейсів.

1.2.2 Недоліки автоматизації

Незважаючи на те, що автоматизація тестування має значний ряд переваг, проте вона має і свої недоліки. Перш за все потрібно розуміти, що автоматизація тестування не відбувається сама по собі, а потребує значних зусиль для створення гарного набору автоматизованих тестів (рисунок 1).

Рисунок 1 – Співвідношення часу розробки та виконання тест-кейсів у ручному та автоматизованому тестуванні

Проте, на жаль, це не єдина складність, з якою пов’язаний ряд значних недоліків та ризиків. Наприклад, технічна кваліфікація співробітників, що займаються автоматизацією тестування, повинна бути значно вищою ніж в їхніх колег, які займаються ручним тестуванням.

Розробка та підтримка автоматизованих тестів, а також необхідної для них інфраструктури, відбирає дуже багато часу і зусиль. До того ж у разі значних змін на проєкті більшість таких тестів доведеться переписувати, а це колосальний об’єм роботи.

Комерційні засоби автоматизації є досить дорогими, а безкоштовні аналоги зазвичай не дозволяють ефективно вирішувати поставлені задачі. Перед проєктом постає питання вибору відповідних засобів, а він є неоднозначним. Проблему вибору ускладнює те, що може виникнути необхідність навчання персоналу та прийняття на роботу нових фахівців. Саме тому автоматизація потребує більш ретельного планування та управління ризиками.

Існують спеціальні підходи за оцінкою придатності та ефективності автоматизованого тестування:

– чим більша різниця між затратами часу на ручне виконання тест-кейсів та на виконання тих самих тест-кейсів автоматизовано, тим більш вигідною є автоматизація;

– чим більша кількість повторів виконання одних і тих самих тест-кейсів, тим більше часу можна зекономити за рахунок автоматизації;

– найскладніше оцінити затрати часу на налагодження, оновлення та підтримку автоматизованих тест-кейсів.

Останній параметр є найбільшою загрозою успіху автоматизації, саме тому для проведення оцінки потрібно залучати найбільш досвідчених спеціалістів.

1.2.3 Області застосування автоматизації

Розглянемо випадки найбільшої застосовності автоматизації тестування.

Регресійне тестування. Необхідність виконувати вручну тести, кількість яких зростає з кожною наступною збіркою, а сенс яких зводиться до перевірки того факту, що функціональність, що коректно працювала до внесення змін, продовжує працювати коректно.

Інсталяційне тестування і налаштування тестового оточення. Безліч рутинних операцій по перевірці роботи інсталятора, розміщення файлів у файловій системі, вмісту конфігураційних файлів і тому подібного. Підготовка додатка в заданому середовищі із вказаними параметрами для проведення основного тестування.

Конфігураційне тестування та тестування сумісності. Виконання одних і тих же тест-кейсів на великій множині вхідних даних, під різними платформами та в різних умовах.

Використання комбінаторних технік тестування. Генерація комбінацій значень і багаторазове виконання тест-кейсів з використанням згенерованих комбінацій в якості вхідних даних.

Модульне тестування. Перевірка коректності роботи атомарних ділянок коду і елементарних взаємодій між ним − нездійсненне для людини завдання за умови, що потрібно виконати тисячі таких перевірок і ніде не допустити помилку.

Інтеграційне тестування. Глибока перевірка взаємодії компонентів в ситуації, коли всі процеси, що піддаються тестуванню, проходять на глибших рівнях, ніж користувацький інтерфейс.

Тестування безпеки. Необхідність перевірки прав доступу, паролів за замовчуванням, відкритих портів, слабких місць поточних версій програмного забезпечення і тому подібного.

Тестування продуктивності. Створення навантаження з інтенсивністю і точністю, недоступною людині. Збір з високою швидкістю великого набору параметрів роботи програми. Аналіз великого обсягу даних з журналів роботи системи автоматизації.

Димовий тест для великих систем. Виконання при отриманні кожної збірки великої кількості досить простих для автоматизації тест-кейсів.

Додатки без графічного інтерфейсу. Перевірка консольних додатків на великих наборах значень параметрів командного рядка (і їх комбінацій). Перевірка додатків і їх компонентів, взагалі не призначених для взаємодії з людиною (веб-сервіси, сервери, бібліотеки тощо).

Рутинні, виснажливі для людини операції. Перевірки, що вимагають порівняння великих обсягів даних, високої точності обчислень, обробки великої кількості розміщених по всьому дереву каталогів файлів, тощо. Особливо, коли такі перевірки повторюються дуже часто.

Перевірка «внутрішньої функціональності» веб-додатків (посилань, доступності сторінок, тощо). Наприклад, перевірити всі 10000+ посилань на предмет того, що всі вони ведуть на реально існуючі сторінки). Автоматизація тут спрощується в силу стандартності завдання − існує багато готових рішень.

Стандартна, однотипна для багатьох проєктів функціональність. Навіть висока складність при первинній автоматизації у такому випадку окупиться за рахунок простоти багаторазового використання отриманих рішень у різних проєктах.

«Технічні завдання». Перевірки коректності протоколювання, роботи з базами даних, коректності пошуку, файлових операцій, коректності форматів і вмісту документів, що генеруються програмою.

Незважаючи на те, що автоматизація тестування застосовна для багатьох випадків, буває й так, що вона може лише погіршити ситуацію. Загалом це стосується тих областей, які потребують людського мислення та переліку технологічних областей. Наприклад, планування, розробка тест-кейсів, тестування зручності для користувача. Також недоцільно застосовувати автоматизацію у випадках, коли буде потрібно перевірити функціональність всього декілька разів, оскільки затрати, витрачені на написання автоматизованих тестів, не окупляться. За умови нестабільності вимог також краще не застосовувати автоматизацію, оскільки існує велика ймовірність того, що тести доведеться переписувати.

Загалом, автоматизація не завжди має грандіозний ефект. Лише при розумному застосуванні можна отримати значну вигоду, і то не відразу. При неправильному чи недоцільному застосуванні автоматизації вона приведе лише до вагомих втрат та розчарування.

1.2.4 Технології автоматизації тестування

Розпочнемо з функціональної декомпозиції − основоположного підходу, який застосовується в більшості технологій автоматизації.

Функціональна декомпозиція [1] − процес визначення функції через її розділення на декілька низькорівневих підфункцій. Вона використовується для спрощення розв’язання поставлених задач та отримання можливості перевикористання фрагментів коду для вирішення різних високорівневих задач.

Розглянемо даний процес на прикладі авторизації в системі (рисунок 2).

Рисунок 2 − Приклад функціональної декомпозиції

Як можна зрозуміти з рисунка, частина низькорівневих дій (пошук та заповнення полів, пошук та натискання кнопок) універсальна та може бути перевикористана під час вирішення інших задач. Це може бути реєстрація на сайті чи оформлення замовлення.

Завдяки даному підходу нема необхідності самостійно реалізовувати дії на найнижчому рівні, оскільки вони вже вирішені авторами відповідних бібліотек та фреймворків.

Розглянемо основні технології автоматизації тестування.

Часткові рішення використовуються у випадках, коли виникає унікальна задача, для вирішення якої нема необхідності використовувати потужні

інструментальні засоби, а достатньо написати нескладну програму на високорівневій мові програмування.

Тестування під управлінням даними використовується для того, щоб підготовити список файлів і передати їх на обробку у випадку, коли в командних файлах повторюються рядки, які виконують одну і ту саму дію над набором файлів. Дана технологія також застосовується для перевірки автоматизації та прав доступу на великому наборі даних імен користувачів та паролів, багаторазового заповнення полів форм різними даними, перевірки реакції додатка, виконання тест-кейсів на основі даних, отриманих за допомогою комбінаторних технік. Дані можуть братися із файлів, баз даних або навіть генеруватися в процесі виконання програми.

Тестування під управлінням ключовими словами є логічним розвитком ідеї про винесення із тест-кейса даних, а саме, винесення зі тест-кейса команд (опису дій). Гарним прикладом інструментального засобу автоматизації тестування, що відповідає даній технології тестування, є Selenium, в якому кожна дія тест-кейса описується у вигляді дії та набору параметрів.

Фреймворками автоматизації тестування є рішення, що виявились успішними та набули популярності. Вони поєднують у собі найкращі якості тестування під управлінням даними, ключовими словами та можливості реалізації часткових рішень на високому рівні абстракції.

Технологія запису та відтворення стала актуальною з появою доволі складних засобів автоматизації, що забезпечують глибоку взаємодію із додатками, що тестуються і операційною системою. Незважаючи на складність внутрішньої реалізації, дана технологія дуже проста у використанні, тому її використовують для навчання початківців. Зауважимо, що технологія запису і відтворення не є універсальним засобом і не може замінити ручну роботу по написанню коду автоматизованих тест-кейсів, але в окремих ситуаціях може пришвидшити вирішення простих рутинних задач.

Тестування під впливом поведінки робить акцент не на окремих технічних деталях, а на загальній працездатності додатка при вирішенні типових користувацьких задач. В основі підходу лежить дуже проста формула «given-when-then»:

– given («за умови») описує початкову ситуацію, в якій знаходиться користувач в контексті роботи з додатком;

– when («коли») описує набір дій користувача в даній ситуації;

– then («тоді») описує очікувану поведінку програми.

РОЗДІЛ 2. ІНСТРУМЕНТИ ТА ПРАКТИЧНІ ПІДХОДИ ДО АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ

2.1 Інструменти автоматизації тестування

У даному підрозділі буде розглянуто найпопулярніші інструменти автоматизації тестування та проведено їхню порівняльну характеристику. В результаті аналізу буде визначено інструмент автоматизації, за допомогою якого буде реалізована практична частина даної роботи.

Selenium [2] є одним з найпопулярніших фреймворків для автоматизації тестування сайтів та веб-додатків. Має відкритий вихідний код і чудово підходить для автоматизаторів із гарними знаннями програмування, оскільки дозволяє самостійно писати скрипти. Хоча Selenium був розроблений ще у 2004 році, але останнім часом він активно розвивається. Дана платформа для автоматизації тестування підтримується різними операційними системами (Microsoft Windows, Mac OS X, Linux) та більшістю популярних веб-браузерів (Google Chrome, Firefox, Opera, Safari, Internet Explorer, Mozilla Suite та інші). Скрипти для даного фреймворка можна писати на Java, C#, JavaScript, Python, Ruby, PHP, Haskell та інших мовах програмування. Перевагами Selenium є можливість написання складних скриптів для автоматизації, а недоліком – досить висока кваліфікація тестувальника для їх написання.

Unified Functional Testing (UTF, в перекладі, комплексне функціональне рішення для тестування) [3] – це набір функцій, призначених для тестування веб-сервісів, сайтів, API та графічного інтерфейсу веб-додатків майже на всіх існуючих платформах.

Даний інструмент має розширений функціонал розпізнавання об’єктів на основі їх зображень, а також гарну документацію по автоматизації.

Katalon Studio [4] – ефективний інструмент для автоматизації процесу тестування сайтів, веб-сервісів, мобільних додатків. Його вважають послідовником Selenium, тому що Katalon Studio запозичив ряд переваг, пов’язаних з інтегрованою автоматизацією тестування програмного забезпечення. Даний інструмент може працювати з JIRA, Jenkins, qTest, Git. Також у нього влаштована функція Katalon Analytics, яка дозволяє користувачу отримати звіти щодо процесу тестування. Вони можуть бути оформлені у вигляді графіків, метрик та діаграм. Перевагою даного інструменту є те, що він безкоштовний і сучасний, а недоліком – його досить молодий вік, у результаті чого присутні деякі неточності в програмі, а також частина його функціоналу ще досі знаходиться на стадії розробки.

Watir (Web Application Testing in Ruby) [5] – інструмент для автоматизації тестування, що використовує у своїй роботі бібліотеку Ruby. Даний інструмент є безкоштовний та кросбраузерний. Watir підтримує тестування під управлінням даними. Основним його недоліком є можливість написання скриптів лише на мові Ruby.

TestComplete [6] – це ефективний інструмент автоматизації тестування мобільних, десктопних та веб-додатків. Він підтримує такі скриптові мови програмування: JavaScript, Python, VBScript, Jscript, C++Script, C#Script, DelphiScript та VB. За допомогою TestComplete можна проводити тестування під управлінням даними та ключовими словами. Даний інструмент має зручну функцію запису процесу із можливістю його відтворення у майбутньому. Також відбувається автоматичне виявлення та оновлення елементів користувацького

інтерфейсу після змін. Це дозволяє мінімізувати час, необхідний для підтримки тестових скриптів. Основним недоліком даної програми для нас є те, що вона має платну ліцензію.

Також існує багато інших платних інструментів, наприклад: IBM Rational Functional Tester, Tricentis Tosca, TestPlant eggPlant, Ranorex, Robot framework та інші. Вони мають ряд переваг у порівнянні із своїми безкоштовними аналогами, але потребують фінансових витрат для їх використання.

Для зручності аналізу розглянутих вище інструментів автоматизації тестування скористаємося таблицею 1.

Таблиця 1 − Порівняння інструментів автоматизації тестування

Продукт	Selenium	UTF	Katalon Studio	Watir	TestComplete
Що можна тестувати	Веб-додатки	Веб (GUI та API), мобільні, десктопні, пакетні додатки	Веб (GUI та API), мобільні додатки	Веб-додатки	Веб (GUI та API), мобільні, десктопні додатки
ОС	Windows, Linux, OS X	Windows	Windows, Linux, OS X	Windows, Linux, OS X	Windows
Скриптові мови	Java, C#, JavaScript, Python, Ruby, PHP, Haskell…	VBScript	Java / Groovy	Ruby	JavaScript, Python, VBScript, C++Script, C#Script…
Навички програмування	Потрібні	Не обов’язкові	Не обов’язкові	Потрібні	Не обов’язкові
Легкість встановлення і використання	Потрібні відповідні навички	Потрібен тренінг	Легко	Потрібні відповідні навички	Потрібен тренінг
Ціна	Безкоштовно	Від $2,300 за рік	Безкоштовно	Безкоштовно	Від €5,428 за рік

Рік створення	2004	1998	2015	2008	1999

Після аналізу даних у наведеній вище таблиці можна зробити висновок, що хоч UTF та TestComplete мають ряд своїх переваг, але вони коштують досить дорого, тому підходять лише для комерційного використання в досить великих компаніях.

Щодо Watir, то це досить непоганий варіант, але так як він підтримує лише одну мову програмування, то на фоні конкурентів, які залишилися, значно програє по цій характеристиці.

Залишилось обрати між Katalon Studio та Selenium. По характеристиках перемагає Katalon Studio, оскільки він є більш сучасним і має ширший спектр можливостей. Але це досить новий продукт, що ще не є стабільним, та інформації по ньому в Інтернеті не так вже й багато. Щодо Selenium, то його основним недоліком є вимога до рівня знань програмування користувача, а із переваг − стабільність. Оскільки рівень знань програмування в нашому випадку не є проблемою, то інструментом для автоматизації в рамках даної роботи було обрано Selenium.

2.2 Робота з веб-елементами

Під час написання автоматизованих тестів багато часу іде на пошуки потрібної складової веб-сторінки. Це можуть бути як кнопки, так і інші графічні блоки. Такий пошук може бути досить складний в ситуаціях, коли веб-елемент не має унікального ідентифікатора чи оригінальної назви класу.

Веб-елемент [7] − це оригінальний об’єкт всередині конструкції веб-сторінки, який генерується тоді, коли користувач, наприклад, відкриває сторінку сайту. В конструкціях мови розмітки HTML всі елементи визначаються за допомогою тегів, класів, атрибутів та вмісту. Використовуючи CSS, сутність і зовнішнє відображення елементів можна редагувати, приховувати або

створювати такі елементи. Сучасні мови програмування взаємодіють з веб-елементами за допомогою об'єктної моделі документа (DOM).

Локатор веб-елемента [7] − це спеціальний об’єкт, що може виявляти елементи по певному запиту. Тобто, за допомогою локаторів можна знайти будь-який веб-елемент. Використання локаторів дозволяє взаємодіяти із структурою програмного забезпечення на програмному рівні: формувати попередній закодований варіант пошуку, на основі якого будуть виконуватися відповідні дії із певними веб-елементами. Продукти автоматизації не вивчають веб-сторінку, так як це робить людина, вони виконують пошук по структурі DOM.

Для пошуку веб-елементів треба використовувати DevTools браузера та інтегроване середовище розробки.

Наприклад, в Google Chrome дослідження сторінки в браузері виконується після попереднього натискання правої кнопки миші на потрібний веб-елемент або за допомогою клавіші F12. Щоб відзначити потрібний елемент, потрібно перейти у вкладку елементів. Потім натиснути на іконку миші і перемістити курсор на необхідний елемент. Завдяки цьому можна досліджувати потрібний клас, тег чи атрибут та його батьківські чи дочірні елементи (рисунок 3).

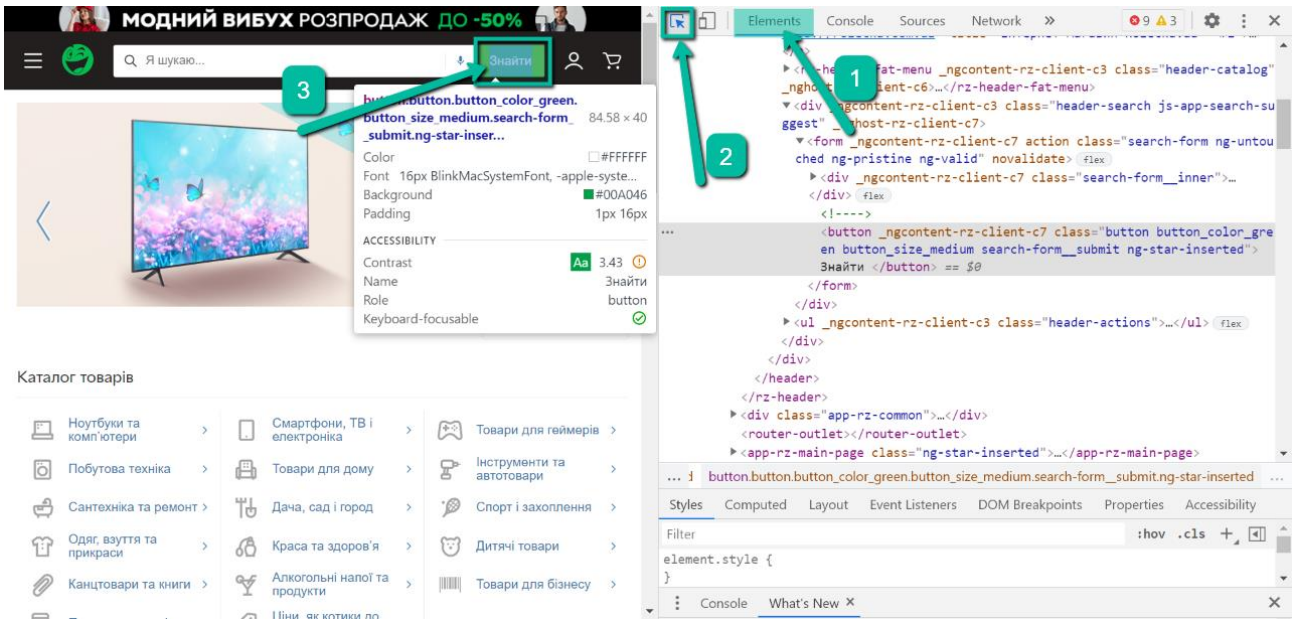

Рисунок 3 − Послідовність кроків для дослідження сторінки в браузері

Дуже важливим етапом при розробці автоматизованих тестів є розробка оригінального запиту для локатора, тому що якщо значення локатора неоднозначне, то у більшості випадків він буде повертати хибнопозитивні параметри. Гарною практикою вважається створювати звичайний стандартний запит, який може якісно ідентифікувати базовий елемент або групу елементів.

Можна орієнтуватися на наступний рейтинг видів запитів по мірі їх спадання:

- ідентифікатор;
- назва;
- назва класу;
- CSS селектор;
- Xpath без тіла тексту;
- текст URL;
- Xpath з текстом.

Оригінальні ідентифікатори і назви класів дозволяють спростити розробку локаторів: всі наступні запити будуть короткими, і не буде необхідності в допоміжних «якорях». У випадках, коли тестувальники можуть впливати на процес розробки, завжди варто просити програмістів створювати виключно оригінальні ідентифікатори для всіх необхідних під час тестування веб-елементів. Але, як показує практика, у більшості випадків елементи не мають унікального ідентифікатора, в результаті чого локатори змушені покладатися на CSS-селектори та Xpath. Для таких випадків існують наступні рекомендації:

- використовувати батьківські веб-елементи в якості якорів, якщо ті мають оригінальними ID;
- уникати Xpath, що містять текст;
- застосовувати параметри «contains», інспектуючи класи в Xpath.

Завжди потрібно перевіряти локатори, оскільки в них можуть бути присутні синтаксичні помилки. В браузері Google Chrome це можна зробити за

допомогою натискання клавіш Ctrl+F на вкладці елементів, у результаті чого відразу відобразиться черга із всіма шуканими елементами.

Автоматизація тестування GUI часто піддається критиці, оскільки тести можуть падати із незрозумілих на перший погляд причин. У більшості випадків ця нестабільність пов’язана з тим, що будь-яка веб-взаємодія викликає стан гонитви. Процес автоматизації та браузер функціонують окремо один від одного, а тестова взаємодія має синхронізуватися із поточним станом сторінки. За умови відсутності такої синхронізації під час виконання автоматизованих тестів будуть виникати помилки, так як система не зможе знайти елементи, що ще не відобразилися у вікні браузера. Дана проблема пов’язана з тим, що автоматизовані тести проходять відповідні кроки значно швидше, ніж би це робила людина, і елементи на сторінці не встигають відобразитися. Найефективнішим методом вирішення даної проблеми є очікування на відображення елемента, перед тим як починати взаємодію із ним. Наприклад, у Selenium WebDriver існує параметр WebDriverWait, за допомогою якого можна змусити програму чекати виконання певних умов перед тим, як переходити до виконання наступних кроків тесту.

2.3 Шаблони автоматизації тестування

Успішність автоматизації зазвичай залежить від правильності впровадження шаблонів тестування, які допомагають покращити надійність та полегшити підтримку автоматизованих тест-кейсів.

Шаблон проєктування автоматизованого тестування [8] − це просте рішення, яке з часом лише підтверджує свою ефективність. Такі рішення вважаються найкращими практиками для проєктів з використанням ООП.

У цьому підрозділі будуть описані найпоширеніші шаблони автоматизації тестування, що використовуються для покращення всієї логіки тестування.

2.3.1 Page Objects

Одним із найпопулярніших підходів до автоматизації тестування є моделювання поведінки частин додатка, що тестується. Такий шаблон втілює один із основних принципів ООП − принцип єдиного обов'язку. Цей принцип гарно демонструється за допомогою наступного прикладу. Під час зміни індивідуального ідентифікатора елемента в коді потрібно буде додати зміни лише в одному місці в коді, а тести, що використовують page objects, автоматично отримають змінене значення без ніяких додаткових змін у коді (рисунок 4).

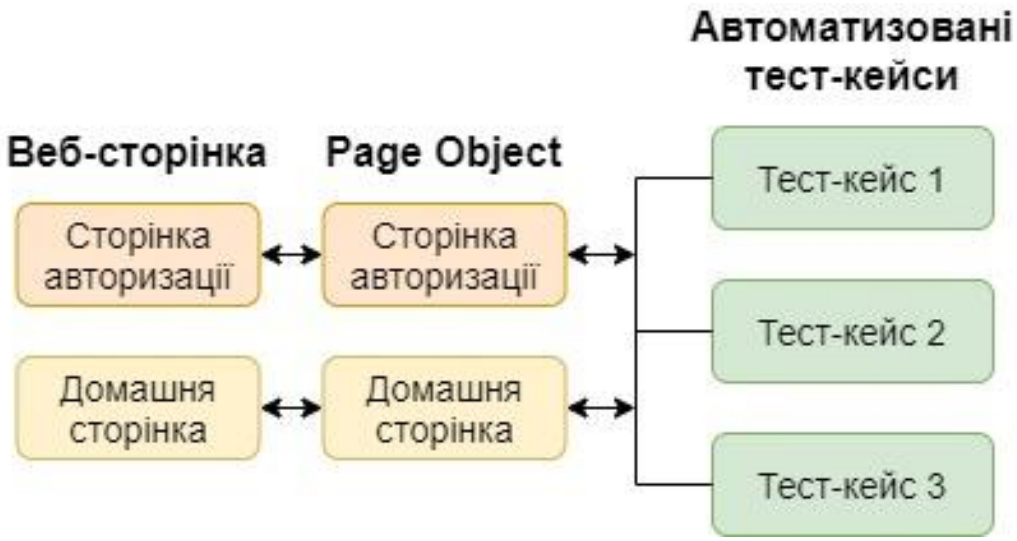

Рисунок 4 − Приклад Page Object

Тобто, основне завдання даного підходу полягає в тому, щоб у випадку зміни запиту для знаходження веб-елемента на сторінці потрібно було поправити код лише в одному місці.

Даний підхід допомагає приховати технічні подробиці HTML та CSS під методами з інтуїтивно зрозумілими назвами. Грамотне найменування методів

допомагає створити простий для розуміння програмний інтерфейс, яким зможе легко навчитися користуватися менш технічно підкований користувач.

Page Objects також зменшує кількість повторень у коді, а це ще один принцип розробки програмного забезпечення − DRY(Don't repeat yourself). Цей принцип означає, що за якусь певну логіку має відповідати лише один шматок коду, і не більше. Дублювання коду ускладнює його підтримку і стабільність, оскільки чим більше змін потрібно додати, тим більша ймовірність того, що програміст допустить помилку, а це може значно погіршити якість кінцевого продукту.

Якщо систематизувати все вище сказане, то можна визначити такі основні переваги використання Page Objects:

– вся логіка сторінки описується в Page Objects класах, а класи, в яких прописуються тест-кейси, використовують публічні методи даного шаблону та перевіряють результат;
– відбувається інкапсуляція;
– локатори можна записати в декларативному стилі.

Отже, можна прийти до висновку, що Page Objects може вирішити більшість проблем під час написання автоматизованих тест-кейсів, хоча він є далеко не єдиним шаблоном, що використовується при автоматизації тестування.

2.3.2 Page Factory

Даний шаблон може застосовуватися не лише до сторінок. Він виник тому, що іноді, щоб форматувати сторінку, необхідно зробити більше дій, ніж просто сказати «нова сторінка» або «відкрити», або щось подібне. Тобто, в цій сторінці прихована ще якась додаткова логіка, і її потрібно кудись зареєструвати, форматувати її елементи і так далі.

У такому випадку потрібно, щоб ця інформація була прихована від того, хто створює сторінку, оскільки вона технічна і неважлива під час написання тест-кейсів.

Даний підхід полягає у тому, що потрібно вказати фабрику Page Factory, і як результат, отримати екземпляр класу цієї сторінки з усіма ініціалізованими елементами, які в ній є.

Додатковий фактор, який буде тут працювати – це ініціалізація всіх елементів, тобто можна буде відкрити будь-яку сторінку відразу, і необов'язково починати з головної сторінки.

2.3.3 Page Elements

Бувають випадки, коли використання Page Objects буде не найкращою ідеєю, це може бути в наступних ситуаціях:

- присутність однакових елементів на багатьох різних сторінках сайту;
- використання особливих елементів при створенні додатка.

Щоб усунути дану проблему можна скористатися унаслідуванням, але найкращим вирішенням буде застосувати агрегацію. Даний підхід втілюється в шаблоні Page Elements [9], що дозволяє розбивати сторінку на менші частини. Завдяки чому ці елементи можна буде використовувати на різних сторінках (рисунок 5).

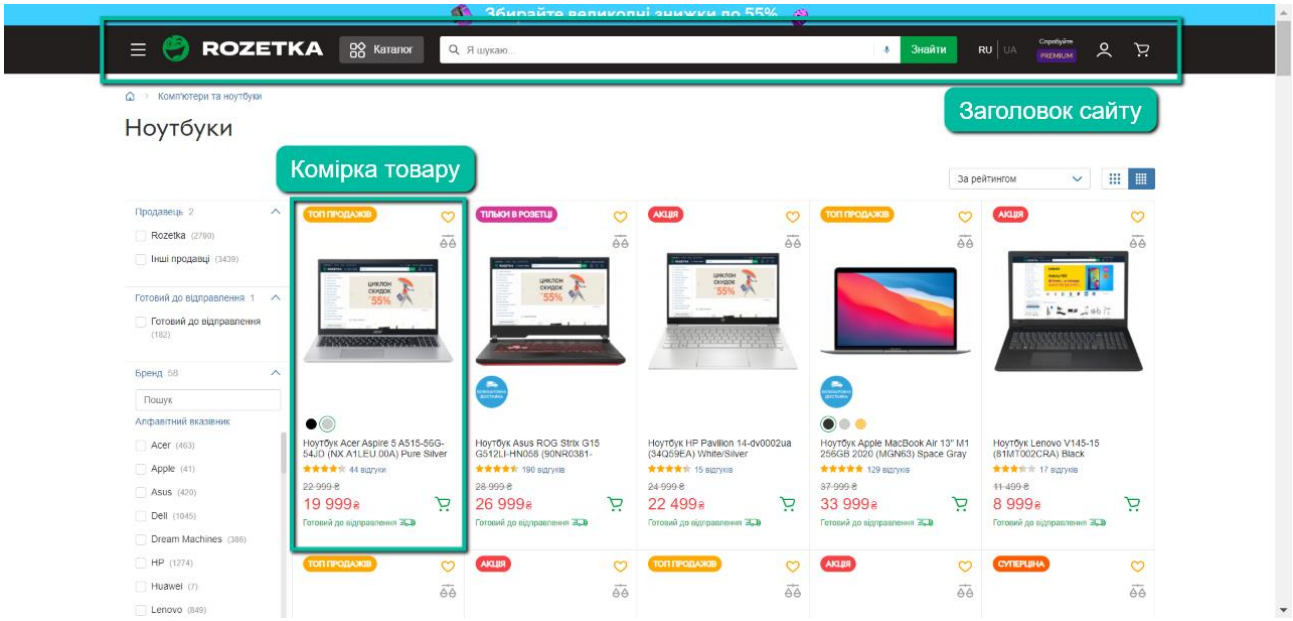

Рисунок 5 – Приклади елементів на сайті

2.3.4 Fluent/Chain of invocations

Даний шаблон застосовується у випадках, коли починає накопичуватися доволі значна кількість сторінок і методів у середині цих сторінок і виникає потреба у визначенні потоку дій. Його суть полягає в тому, щоб щоразу, як автоматизатор захоче обрати наступну дію, з’явилася автопідказка.

Кожного разу, коли виконується якась дія, повертається контекст. Це може бути та сама сторінка, перевантажена сторінка, інша сторінка, логічний компонент чи щось інше. Для того, щоб перервати цей ланцюжок, потрібно застосувати кінцеву термінальну операцію, після якої ланцюжок буде перервано, і наступна дія має виконуватися без підказок.

Реалізується це дуже просто – в Page Objects або інших місцях, де можна застосувати цей шаблон, потрібно використовувати значення, що повертається, яким може бути будь-який об'єкт, з яким далі буде продовжено роботу.

Завдяки цьому шаблону код стає значно компактнішим, з нього прибираються деякі дублікати, а також він стає прозорішим та зрозумілішим.

2.3.5 Screenplay (також відомий як Journey)

Даний шаблон втілює в собі принципи проектування SOLID для автоматизованого тестування. Screenplay [10] розділяє Page Objects на частини, що робить автоматизовані тест-кейси надійнішими та легшими у підтримці. Ще однією суттєвою перевагою є те, що він робить тестові сценарії зрозумілішими за рахунок систематизації тестових сценаріїв в зручні для читання блоки. Цей шаблон використовує ідею акторів, задач та цілей для формального опису тестів, відмовляючись від термінів взаємодії із системою. Тобто, згідно з Screenplay, потрібно описувати тести в термінах актора, у якого є певні цілі.

Актор має можливості, завдяки яким він може виконувати завдання для досягнення цілей. Завдання – це концепції бізнес-рівня, їх цікавить, чи намагаємось ми взаємодіяти із GUI шляхом взаємодії з веб-сервісом. Актори використовують свою здатність взаємодіяти з системою для виконання поставленого завдання. Дії – це класи, які інкапсулюють взаємодію між користувачем та системою. Під час використання шаблону Screenplay потрібно думати не в рамках сторінок чи полів, а з точки зору користувача, що взаємодіє з системою. Актор буде вводити щось в елементи і натискати на них, для цього буде використовуватися Page Object, єдине призначення якого в даній ситуації – це збереження інформації про локатори для поля. Також у даній моделі актори можуть задавати запитання про стан системи: вони використовують такі питання для перевірки своїх припущень та очікувань щодо того, якими будуть отримані результати. Актори мають отримати відповіді на задані питання, на основі яких прийняти рішення, чи було пройдено даний тестовий випадок чи ні (рисунок 6).

Рисунок 6 − Приклад шаблону Screenplay

Хоча, на перший погляд, використання цього шаблону здається досить обтяжливим, але на практиці він заощаджує багато часу за рахунок зменшення затрат на обслуговування та підтримку.

2.3.6 Value Object

Суть даного шаблону така: якщо є кілька об'єктів, об'єднаних логічно між собою, тоді їх можна трансформувати, ввівши додатковий ValueObject [11], який агрегує в собі всю інформацію.

Після того, як він створюється, його не можна змінити, тому що його завдання в тому, щоб слугувати для передачі даних з точки А в точку Б, а не для того, щоб бути модифікованим або нести побічні ефекти.

Щоб спростити створення і роботу таких об’єктів при роботі з Java, можна скористатися інструментом Lombok. Цей інструмент легко дозволить робити компактні елементи, використовуючи тільки гетери для отримання даних, а у випадку, якщо необхідні конструктори, які так само генеруються в будь-якій кількості, необов'язково для цього писати код руками.

2.3.7 Data Registry

Цей шаблон вирішує наступну проблему: припустимо, в тестах використовуються дані і для того, щоб вони були незалежні один від одного, потрібно якимось чином відв'язати їх один від одного, але в результаті можна прийти до того, що виходять залежні тести, які будуть «знати» про логіку один одного. Підхід Data Registry дозволяє генерувати унікальні дані і стежити за їх унікальністю. Найпростіший підхід: використання статичного інкрементного багатопотокового лічильника, який постійно інкрементується на одиницю, тим самим гарантуючи унікальність даних.

Реалізація цього шаблону може бути значно складнішою, наприклад, він може брати дані з бази даних чи з файлу. Суть у тому, що кожен, хто працює зі своїм тестом, перш ніж почати роботу з якимись унікальними даними, запитує у Registry дані, хоча він не знає алгоритму рандомізації, але завжди впевнений, що отримає дані незалежно від інших тестів. Тим самим зростає захищеність тестів від помилок в перетині по даних.

2.3.8 Порти та адаптери

Архітектура портів та адаптерів спрямована на те, щоб переконатися, що використовується принцип єдиної відповідальності, іншими словами, конкретний об'єкт повинен робити щось одне і мати тільки одну причину для зміни. Під час застосування такого шаблону в автоматизації потрібно переконатися, що код автоматизованого тест-кейсу відокремлений від інших частин програми. Цей підхід дає можливість міняти місцями повільні компоненти та швидкі симулятори, чим дозволяє запускати тест-кейс та тестований додаток в одному процесі.

2.3.9 Presenter First

Даний шаблон є модифікацією MVC для організація коду і розробки з метою створення повністю протестованого програмного забезпечення з використанням TDD підходу до розробки. Пояснення Presenter First наступне: якщо намалювати схему для MVC у вигляді блоків та стрілок, то можна побачити, що GUI має чітко визначені зв’язки з моделлю та контролером. Якщо в режимі виконання програми можна замітити їх моделями та контролерами, які створює і контролює автоматизований тест-кейс, тоді можна перевірити, що GUI поводиться некоректно. Також можна налаштувати модель та контролер так, щоб вони імітували всі види некоректної поведінки, наприклад, вихід мережі з ладу.

РОЗДІЛ 3. СТВОРЕННЯ НАБОРУ АВТОМАТИЗОВАНИХ ТЕСТ-КЕЙСІВ

У даному розділі буде описано інструментарій для створення автоматизованих тестів, а також наведено приклади практичного використання підходів та шаблонів, описаних у двох попередніх розділах даної роботи.

3.1 Інструментарій

Для роботи з браузером було вирішено обрати Selenium WebDriver, оскільки він найрозвиненіший в порівнянні зі всіма іншими безкоштовними аналогами. Мовою програмування було обрано Java 11, оскільки ця мова програмування є однією з найпопулярніших на сьогоднішній день.

3.1.1 Selenium WebDriver

Webdriver [12] − це інструмент управління браузером, який максимально правдоподібно імітує дії користувача.

Selenium Webdriver підтримує багато мов програмування (Java, JavaScript, C#, Python, PHP, Ruby, та інші). Це проєкт з відкритим кодом, який має велику кількість користувачів.

Його можна використовувати як для автоматизованого тестування веб-додатків, так і для виконання інших типових завдань, пов’язаних з роботою з веб-браузером. Selenium Webdriver безпосередньо викликає команди браузера, використовуючи його API.

Для роботи з Selenium Webdriver потрібно:

– браузер − це веб-браузер (наприклад, Google Chrome, Firefox, Opera, Safari, Internet Explorer, Mozilla Suite тощо певної версії, встановлений на певній ОС зі своїми налаштуваннями), роботу якого потрібно автоматизувати;

– драйвер −це найважливіша сутність, яка запускає браузер та відправляє йому команди, а вкінці роботи закриває його, у кожного браузера свій драйвер, це пов’язано з тим, що різні браузери мають свої особливі команди управління, які мають різну реалізацію;

– скрипт − це код, який містить набір команд певною мовою програмування для веб-драйвера, він використовується Selenium Webdriver bindings бібліотеками, доступними користувачам на різних мовах програмування;

– веб-елемент − ще одна важлива сутність, що являє собою абстракцію над веб-елементами (кнопки, поля для вводу даних, посилання та інші);

– By −це абстракція над локатором веб-елемента, даний клас інкапсулює інформацію про локатор елемента, тобто всю необхідну інформацію для знаходження відповідного елемента на сторінці.

3.1.2 Java, Maven, Junit

Java [13] − об'єктно-орієнтована мова програмування на основі класів. Вона призначена для того, щоб розробники могли писати код програми один раз і запускати його де завгодно. Це означає, що скомпільований Java код може працювати на всіх платформах, що підтримують Java, без необхідності перекомпіляції. Java програми зазвичай компілюються в байт-код, який може працювати на будь-якій віртуальній машині Java (JVM), незалежно від базової архітектури комп'ютера. Синтаксис в Java С-подібний, але має менше низькорівневих об'єктів. Під час виконання Java забезпечує динамічні можливості (відображення та модифікація коду виконання), які, як правило, недоступні в традиційних компільованих мовах програмування.

Maven [14] − це інструмент автоматизації збірки, який використовується в основному для проєктів Java. Maven втілює в собі два аспекти побудови програмного забезпечення: як програмне забезпечення побудовано та його залежності. На відміну від попередніх інструментів, таких як Apache Ant, він використовує домовленості для процедури побудови, тому потрібно записати лише винятки. XML-файл описує як програмне забезпечення проекту будується, його залежності від інших зовнішніх модулів і компонентів, порядок складання, каталогів і необхідних плагінів. Він поставляється із заздалегідь визначеними цілями для виконання певних, чітко визначених, завдань, таких як компіляція коду та його упаковка. Maven динамічно завантажує бібліотеки Java та плагіни Maven з одного або декількох сховищ, таких як Maven 2 Central Repository, і зберігає їх у локальному кеші. Цей локальний кеш завантажених артефактів можна оновити артефактами, створеними локальними проєктами, також можна оновити загальнодоступні сховища.

Junit [15] − одна з найпопулярніших платформ для тестування в екосистемі Java. Версія JUnit 5 містить низку нововведень, метою яких є підтримка нових функцій Java 8 і новіших версій, а також забезпечення багатьох різних стилів тестування. IntelliJ підтримує JUnit 5 за замовчуванням. Тому запустити JUnit 5 на IntelliJ досить просто, наприклад, за допомогою комбінації клавіш Ctrl+Shift+F10.

JUnit 5 складається з трьох різних під-проектів:

- Platform JUnit;
- JUnit Jupiter;
- JUnit Vintage.

Platform JUnit відповідає за запуск тестових платформ на JVM. Він визначає стабільний та потужний інтерфейс між JUnit та його клієнтом, таким як інструменти побудови. Кінцевою метою є те, як його клієнти легко інтегруються з JUnit у пошуку та виконанні тестів.

JUnit Jupiter включає нові моделі програмування та розширення для написання тестів у JUnit 5.

JUnit Vintage підтримує запуск тестів з JUnit 3 та JUnit 4 на платформі JUnit 5.

3.2 Створення і початкова структура проекту

3.2.1 Створення проекту

Для того, щоб почати писати автотести знадобиться [16]:

- середовище розробки Intellij IDEA;
- встановлені Java та Maven;
- браузер Google Chrome та chromedriver − програма для передачі команд браузеру.

Після створення нового Maven проекту у Pom-файл (файл, що зберігає список усіх бібліотек (залежностей), які використовуються в проекті) потрібно додати дві бібліотеки: Selenium Java та Junit, які можна знайти на репозиторії Maven (рисунок 7).

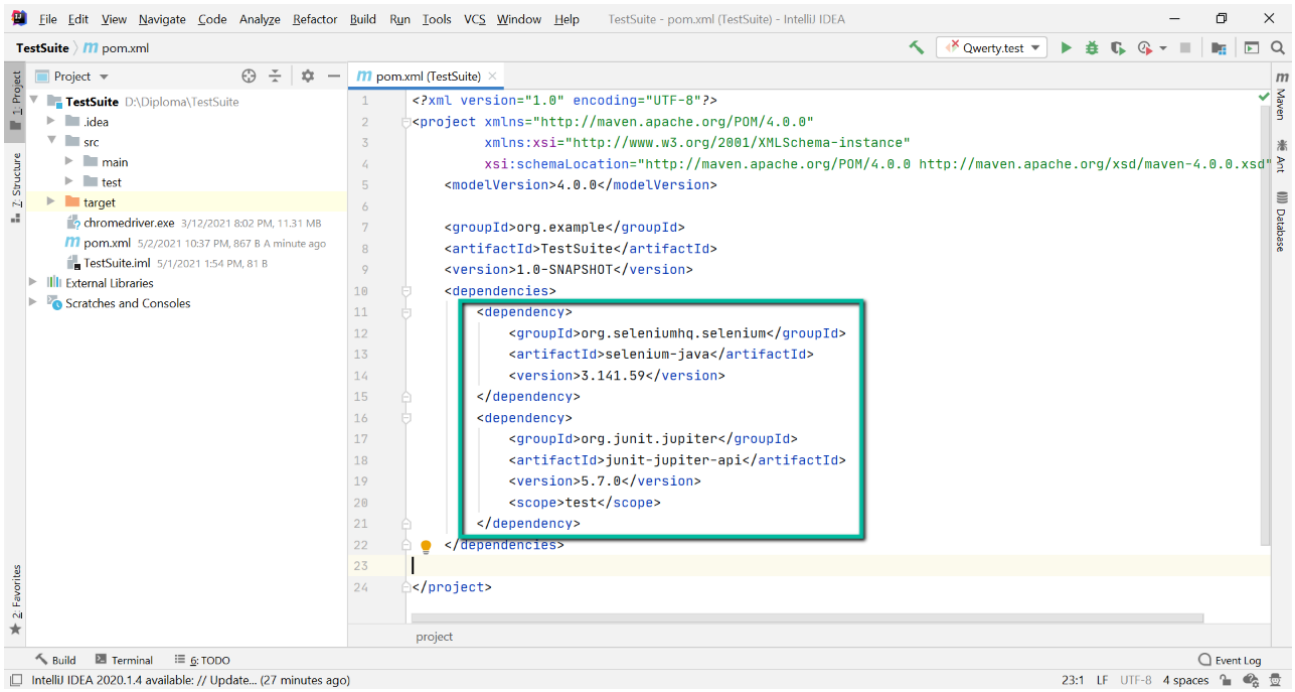

Рисунок 7 – Бібліотеки додані до Pom-файлу

3.2.2 Структура проекту

В класі Tests буде описана логіка тесту. Створимо в цьому класі метод «setUp()», в якому будуть описані попередні налаштування. Отже, для запуску браузера необхідно створити об'єкт драйвера:

```
WebDriver driver = new ChromeDriver();
```

Перед створенням об'єкта WebDriver слід встановити залежність, що визначає шлях до chomedriver (в ОС сімейства Windows додатково необхідно вказувати розширення .exe):

```
System.setProperty("webdriver.chrome.driver",
"D://Diploma/TestSuite/chromedriver.exe");
```

Щоб хід тесту відображався в повністю відкритому вікні, необхідно сказати про це драйверу:

```
driver.manage().window().maximize();
```

Трапляється так, що елементи на сторінках доступні не відразу, і необхідно дочекатися появи елемента. Для цього існують очікування. Вони бувають двох видів: явні і неявні. У прикладі буде використано неявне очікування Implicitly Wait, яке задається спочатку тесту і буде працювати при кожному виклику методу пошуку елемента:

```
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
```

Таким чином, якщо елемент не знайдений, то драйвер буде чекати його появи протягом заданого часу (10 секунд). Як тільки елемент буде знайдений, драйвер продовжить роботу, однак, в іншому випадку тест впаде по закінченню часу.

Для передачі драйверу адреси сторінки використовується команда:

```
driver.get("https://rozetka.com.ua/");
```

На даному етапі наш метод буде мати наступний вигляд (рисунок 8):

```
public static void setUp() throws Exception {
    System.setProperty("webdriver.chrome.driver", "D://Diploma/TestSuite/chromedriver.exe");
    WebDriver driver = new ChromeDriver();
    driver.manage().window().maximize();

    driver.manage().timeouts().implicitlyWait( time: 10, TimeUnit.SECONDS);

    driver.get("https://rozetka.com.ua/");
}
```

Рисунок 8 – Метод із попередніми налаштуваннями

Тепер для зручності винесемо назву сторінки в окремий файл (а трохи пізніше і деякі інші параметри).

Створимо в каталозі «test» ще один каталог з назвою «resources», а в ньому звичайний файл «conf.properties», в який помістимо змінну homepage та шлях до драйвера (рисунок 9).

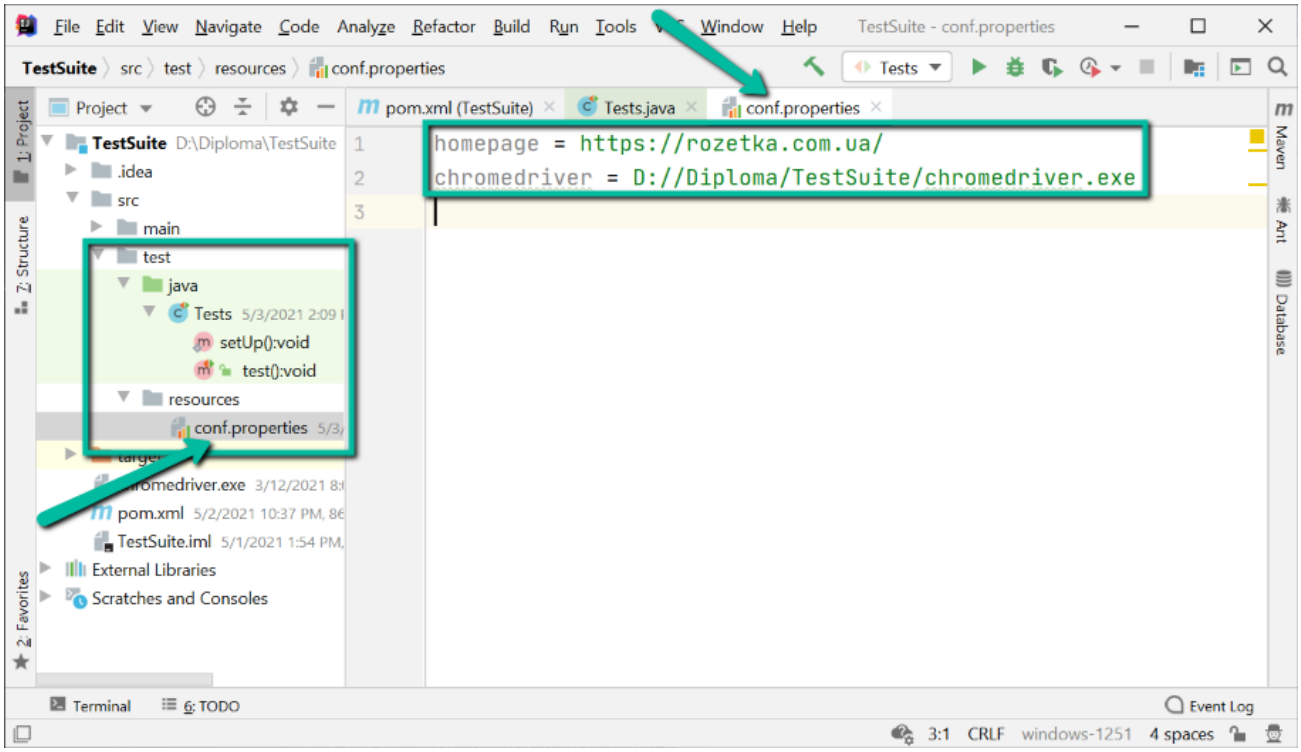

Рисунок 9 – Початковий вміст файлу «conf.properties»

В папці Java створимо ще один клас «ConfProperties», який буде читати записані в файл «conf.properties» значення (рисунок 10).

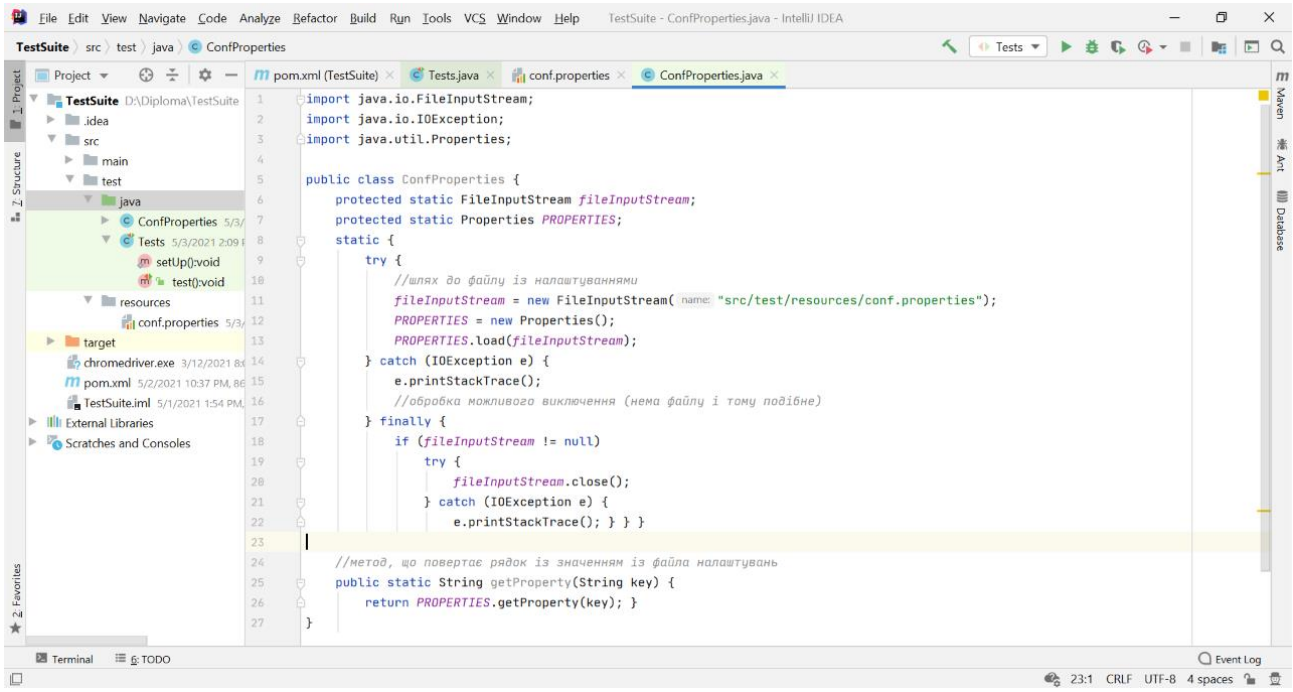

Рисунок 10 – Вміст класу «ConfProperties»

Тепер повернемося до нашого тест-кейсу, змінимо у ньому шлях та посилання. Метод «setup()» позначимо анотацією Junit5 «@BeforeAll», яка вказує на те, що метод буде виконуватися один раз до виконання всіх тестів в класі. Тестові методи в Junit позначаються анотацією @Test (рисунок 11).

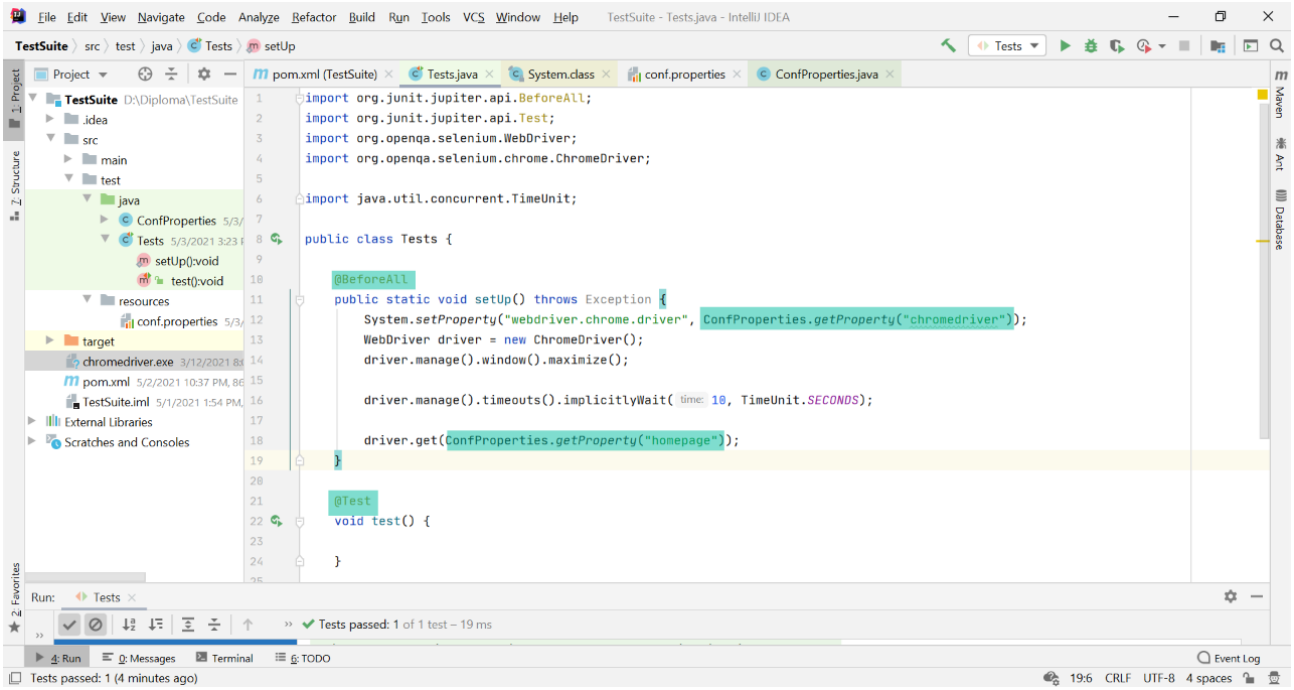

Рисунок 11 – Початковий вміст файлу із тест-кейсами

3.3 Застосування шаблонів

У даному підрозділі будуть описані приклади застосування шаблонів, описаних у попередньому розділі даної роботи.

3.3.1 Page Elements

При використанні Page Elements елементи сторінок, а також методи безпосередньої взаємодії з ними, виносяться в окремий клас.

Для детальної демонстрації написання автоматизованого тест-кейсу візьмемо за приклад наступну послідовність дій:

– користувач відкриває інтернет-магазин;
– користувач вводить пошуковий запит та натискає кнопку «Знайти»;
– відбулася зміна посилання.

Тест-кейс вважається успішно пройденим, якщо в посиланні після натиску на кнопку «Знайти» з’явиться слово, що було введене під час пошуку.

В рамках даної дипломної роботи як приклад веб-застосунку буде використано інтернет-магазин «ROZETKA».

Створимо в папці java клас HeaderElements, який буде містити локатори елементів заголовка інтернет-магазину і методи для взаємодії з ними (рисунок 12).

Відкриємо головну сторінку інтернет-магазину (https://rozetka.com.ua) в браузері Google Chrome. Для визначення локаторів елементів, з якими буде взаємодіяти автотест, скористаємося інструментами розробника. Визначимо локатори для елементів, необхідні для проходження даного тест-кейсу.

Для локаторів елементів в Page Element використовується анотація @FindBy. Для того, щоб анотація @FindBy запрацювала, необхідно використовувати клас PageFactory. Для цього створимо конструктор і передамо йому в якості параметра об'єкт Webdriver.

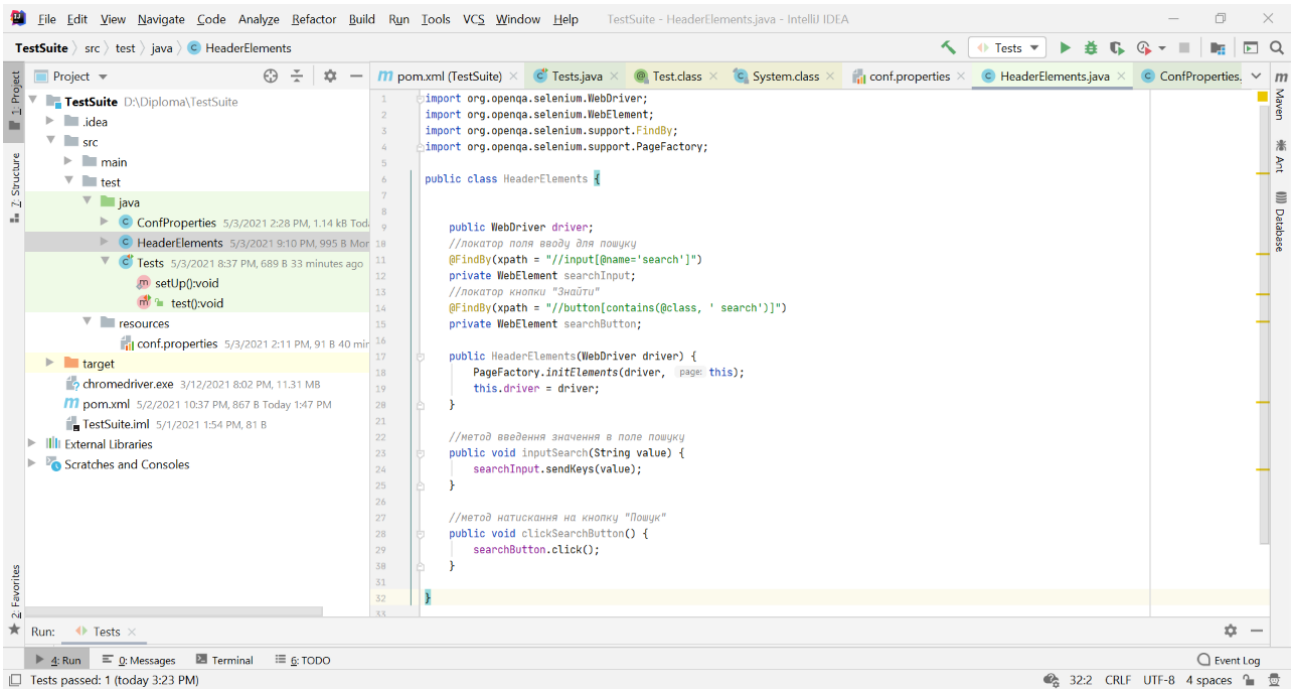

Рисунок 12 – Вміст класу HeaderElements

Повернемося до класу Tests і додамо до нього клас HeaderElements шляхом оголошення статичних змінних з відповідними іменами, а також сюди ж винесемо змінну для драйвера. В анотації @BeforeAll створюємо екземпляр класу HeaderElements і дамо посилання на нього. Створення екземпляра відбувається за допомогою оператора new. Як параметр вказуємо створений перед цим об'єкт driver, який передається конструктору класу, створеному раніше. Також зміниться опис створення екземпляра драйвера, тому що він буде оголошений в якості змінної (рисунок 13).

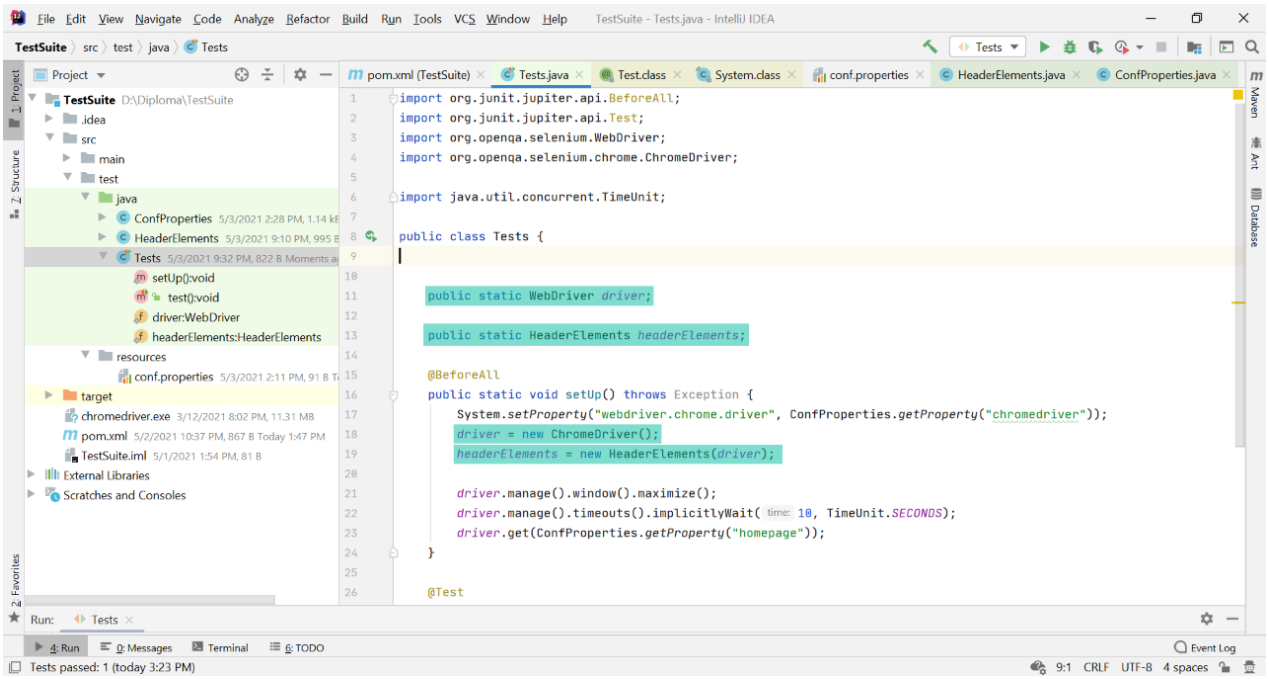

Рисунок 13 – Віст класу Tests

Метод, що перевірятиме вміст URL на наявність в ньому пошукового слова матиме наступний вигляд (рисунок 14):

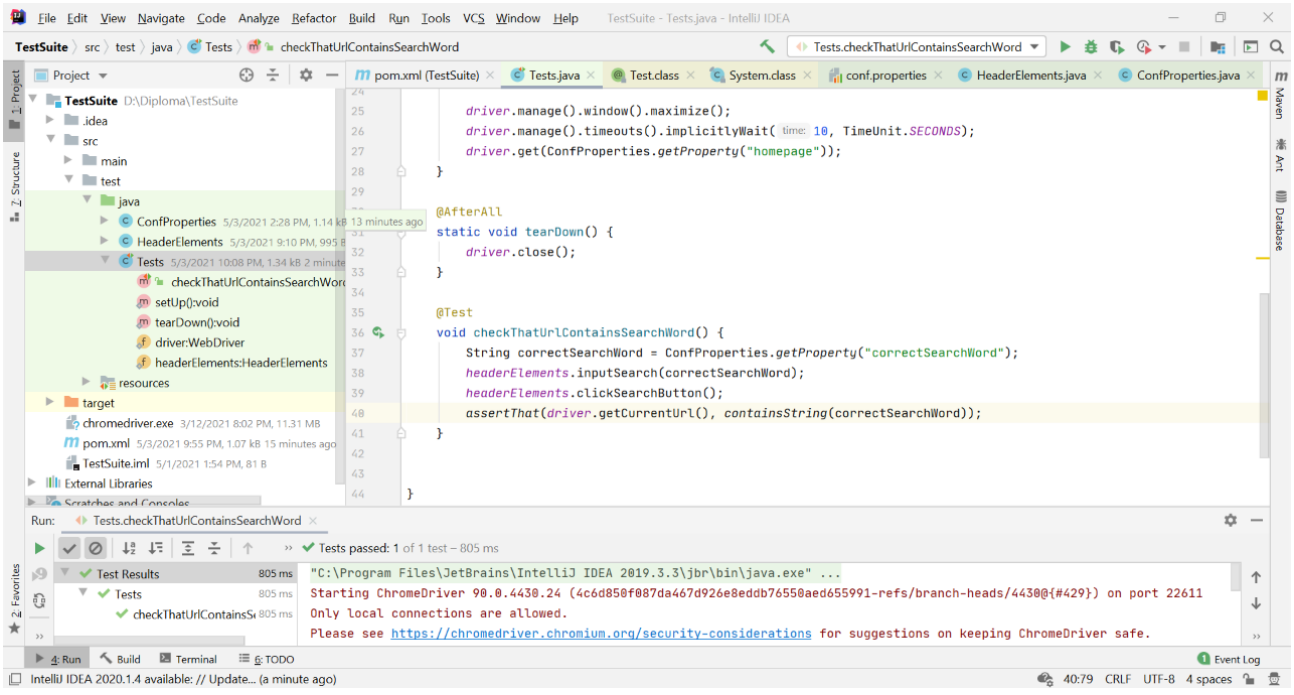

Рисунок 14 – Реалізація методу перевірки вмісту URL після пошуку

Також було створено фінальний метод і позначено його анотацією @After (методи, помічені цієї анотацією, виконуються один раз після завершення всіх тестових методів класу). У цьому методі здійснюється закриття вікна браузера.

Як можна побачити із рисунка 14 описаний вище тест-кейс виконався успішно.

3.3.2 Page Object

Аналогічно до шаблону проєктування, описаного у попередньому підрозділі, в рамках практичної частини даної дипломної роботи також був реалізований шаблон Page Object для сторінок кошика та результатів пошуку.

Розглянемо детальніше клас CartPage як приклад реалізації шаблону Page Object для сторінки кошика інтернет-магазину «ROZETKA». У даному класі містяться локатори для всіх необхідних для тестування веб-елементів (рисунок 15), а також методи, за допомогою яких організовано доступ до інформації, яку можна отримати з цих локаторів чи дії над веб-елементами, представленими даними локаторами.

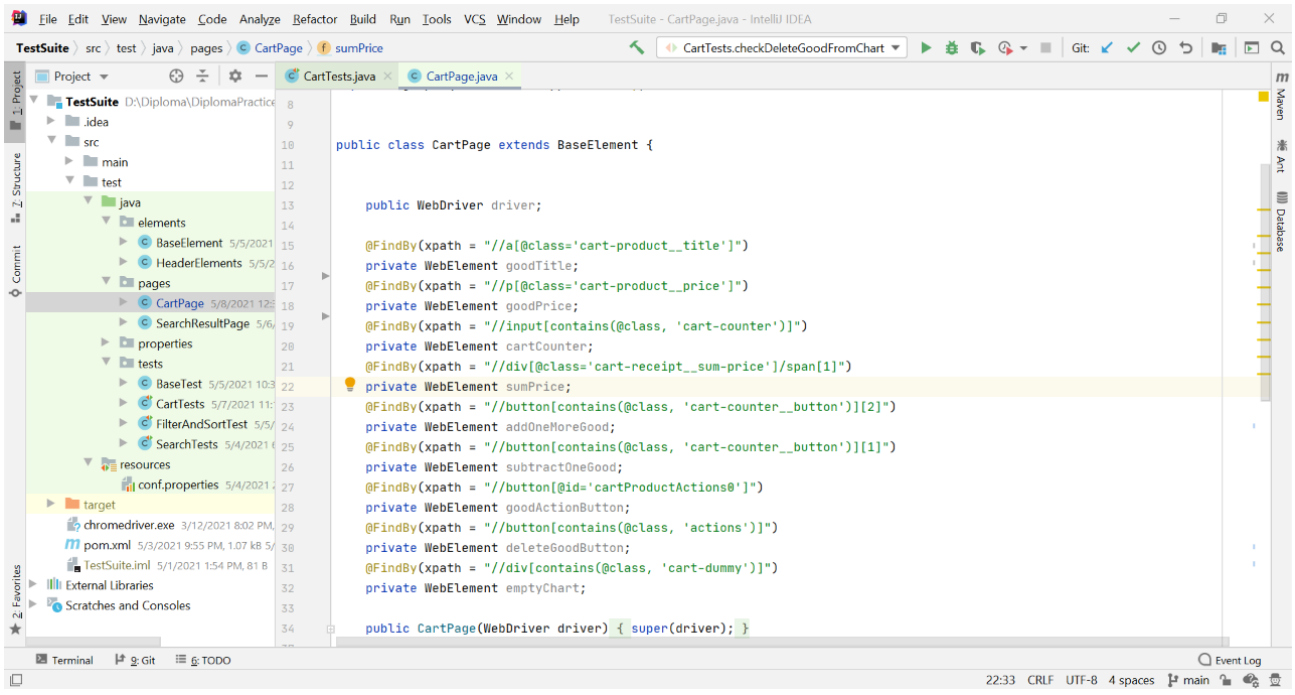

Рисунок 15 – Локатори класу CartPage

Наприклад, в класі CartPage містяться локатори наступних веб-елементів (рисунок 16):

– назва товару;

– ціна товару;
– кількість товару;
– сума замовлення;
– кнопка додавання товару;
– кнопка віднімання товару;
– кнопка, що відкриває можливості дій із товаром;
– кнопка видалення товару із кошика;
– індикатор того, що корзина порожня.

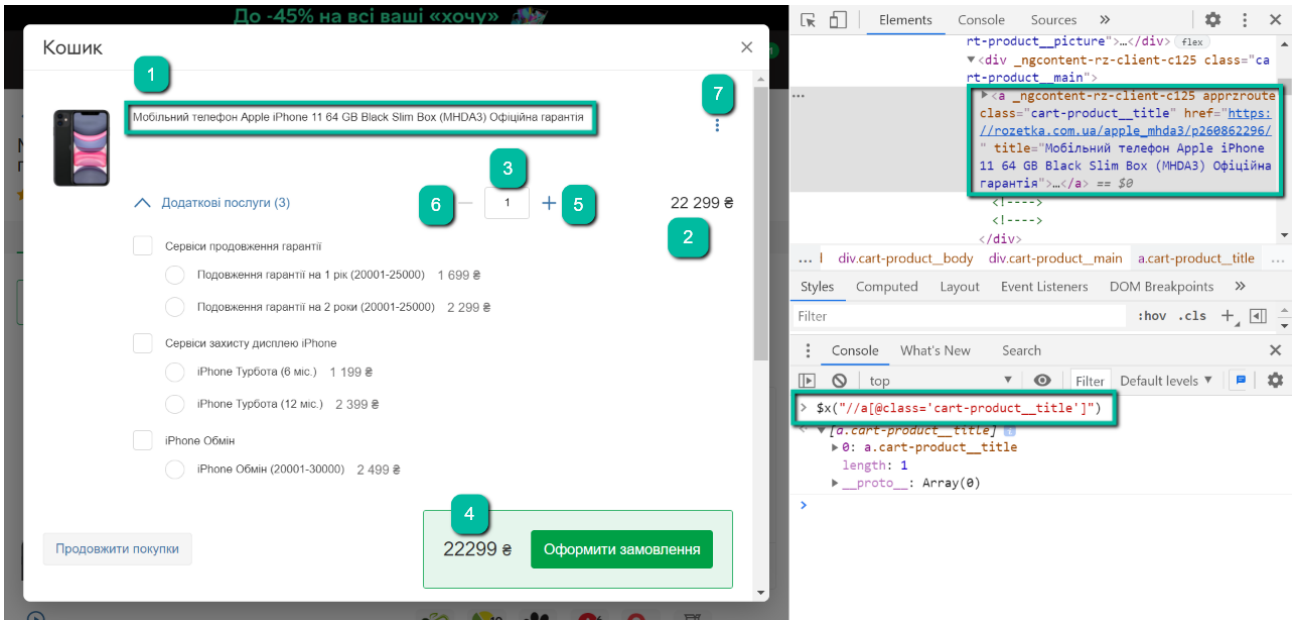

Рисунок 16 – Приклад пошуку локатора для назви товару та вигляд веб-елементів, для яких були написані локатори у класі CartPage

Клас CartPage містить наступні методи доступу до інформації з перелічених вище веб-елементів: це може бути отримання ціни чи назви товару, натискання на клавіші, очікування на відображення елемента на сторінці і тому подібне (рисунок 17).

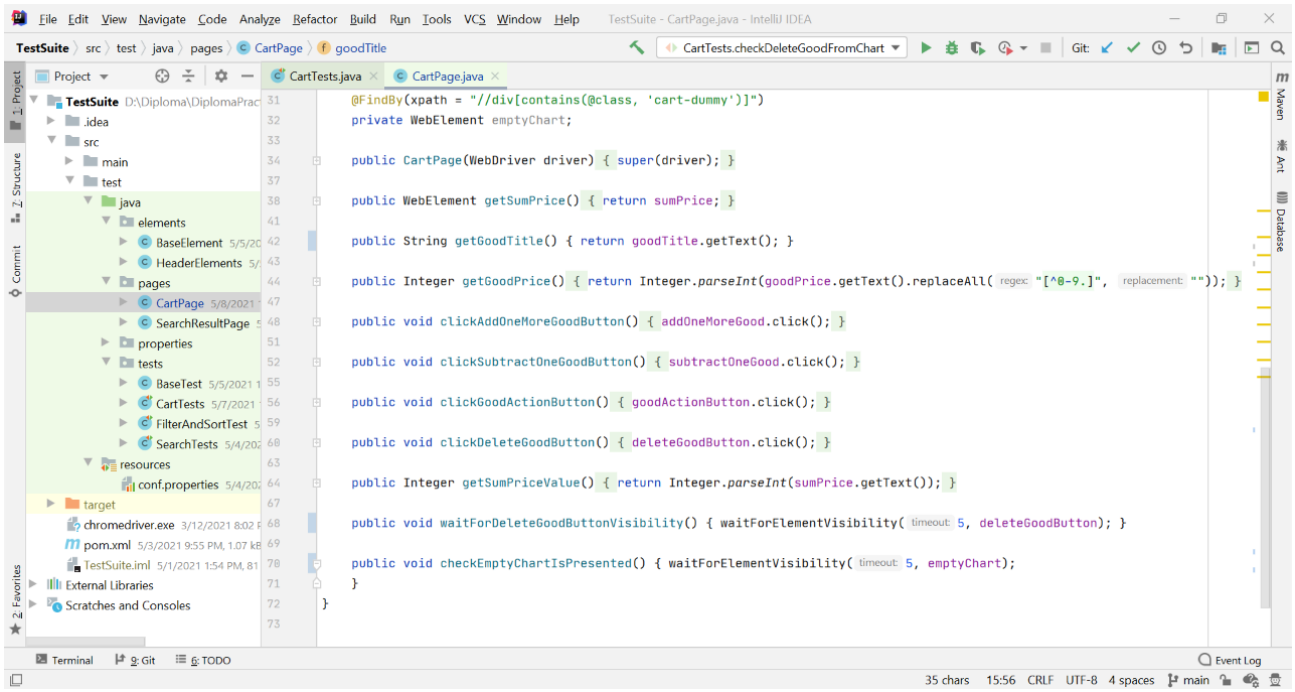

Рисунок 17 – Приклад методів реалізованих в класі CartPage

3.4 Остаточна структура проекту

У процесі виконання практичної частини дипломної роботи було зроблено деякі структурні зміни. Вони були необхідні для зручності та кращого вигляду проекту [17].

По-перше, було відділено реалізацію шаблонів Page Element та Page Objects (логіка доступу до веб-елементів) від логіки тест-кейсів. Це було реалізовано шляхом розбиття відповідних файлів по пакетах elements, pages та tests. По-друге, було створено два додаткових класи: BaseElement та BaseTest.

Клас BaseElement є базовий для класів, що є реалізацією шаблонів Page Object та Page Element. У даному класі описані базові методи взаємодії з елементами, такі як: очікування на завантаження сторінки, очікування на видимість елемента, відображення певного тексту на елементі чи зміна тексту, а

також скрол до елемента (рисунок 18). Від класу BaseElement унаслідуються всі класи елементів і сторінок.

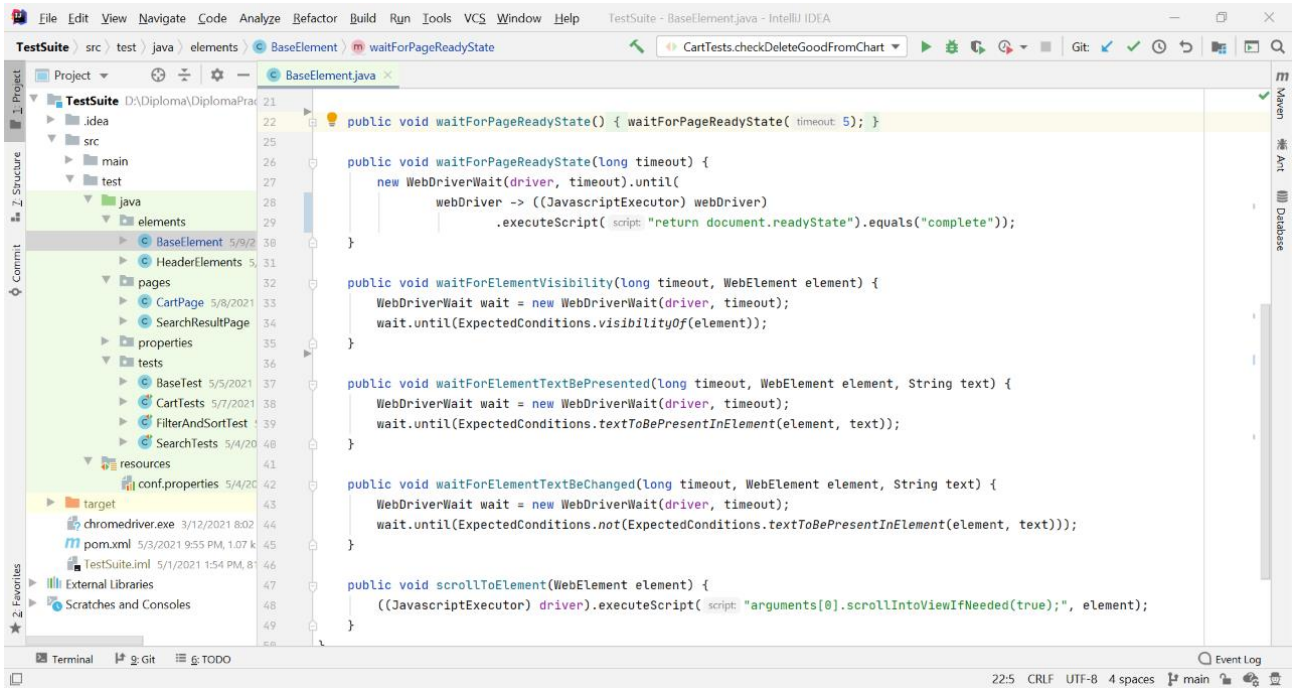

Рисунок 18 – Методи класу BaseElement

Клас BaseTest є батьківським для всіх класів із тестами, в ньому реалізовані методи, що виконуються як перед всім набором тест-кейсів, так і перед кожним окремим тест-кейсом і після нього. Також в даному класі присутні методи отримання новоствореного об’єкта, нової сторінки чи елемента кожного із типів, описаних вище та отримання поточного URL веб-браузера (рисунок 19).

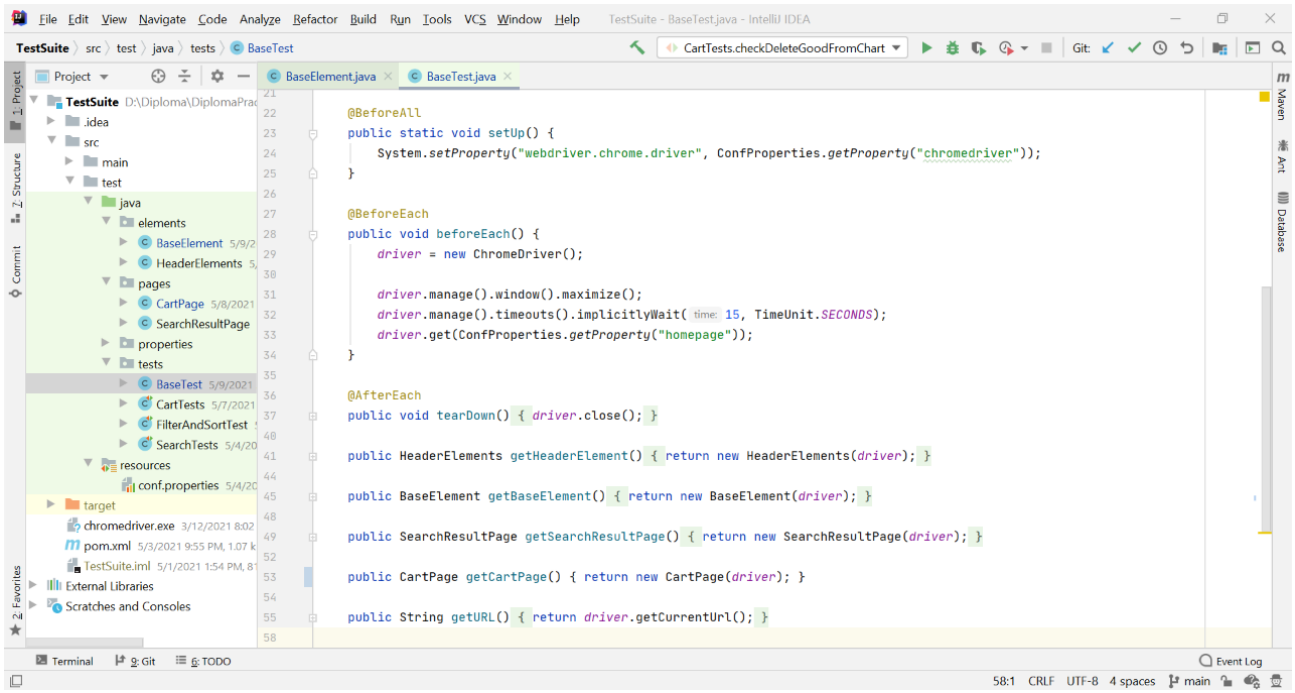

Рисунок 19 – Методи класу BaseTest

3.5 Результати написання набору тест-кейсів

У рамках даної дипломної роботи було створено набір автоматизованих тест-кейсів, що покривають основну функціональність інтернет-магазину «ROZETKA». За кожну функціональність відповідає окремий тест-клас, а саме:

– пошук (SearchTests);
– фільтрація і сортування результатів пошуку (FilterAndSortTest);
– кошик (CartTests).

У наступних підрозділах буде наведено детальніший опис кожного із них.

3.5.1 Тестування пошуку

У класі SearchTests містяться тест-кейси, які покривають функціональність пошуку на сайті. Ось перелік даних тест-кейсів:

– checkThatUrlContainsSearchWord перевіряє те, чи після вводу пошукового слова і натискання на кнопку «Знайти» відбулася зміна URL (пошукове слово має мітитися в ньому);

– checkThatSearchResultsContainsSearchWord перевіряє те, чи слово пошуку міститься у всіх заголовках результатів пошуку;

– checkThatSearchWithWrongSearchWord є негативним тест-кейсом, який перевіряє те, що з’явиться відповідне повідомлення про те, що за заданими параметрами не було знайдено жодного товару.

3.5.2 Тестування фільтрації та сортування результатів пошуку

У класі FilterAndSortTest містяться тест-кейси, що покривають функціональність фільтрації та сортування результатів пошуку. Нижче знаходиться перелік даних тест-кейсів:

– checkMinPriceFilter перевіряє коректність фільтрації результатів пошуку після визначення конкретного значення мінімальної ціни за товар;

– checkMaxPriceFilter перевіряє коректність фільтрації результатів пошуку після визначення конкретного значення максимальної ціни за товар;

– checkCheapFirstSortOption перевіряє коректність сортування результатів пошуку за зростанням ціни;

– checkExpensiveFirstSortOption перевіряє коректність сортування результатів пошуку за спаданням ціни;

– checkMemoryFilter перевіряє коректність фільтрації результатів пошуку за обраним значенням параметра, що відповідає розміру пам’яті шуканого товару.

3.5.3 Тестування роботи з кошиком

У класі CartTests містяться тест-кейси, що покривають функціональність роботи з кошиком. Перелік даних тест-кейсів наступний:

– checkAddToCartHeaderCounter перевіряє зміну лічильника кошика на заголовку сайту після додавання товару до кошика;

– checkItemInCart перевіряє відповідність ціни і назви доданого до кошика товару;

– checkSumAfterChangingQuantityOfGoods перевіряє суму замовлення після збільшення та зменшення кількості певного товару в кошику;

– checkAddDifferentGoodsToChartsAndCheckPrice перевіряє можливість додавання різних товарів до кошика та коректності підрахунку суми замовлення при цьому;

– checkDeleteGoodFromChart перевіряє можливість видалення товару із кошика.

3.5.4 Перевірка стабільності набору тест-кейсів

Тести для автоматизації тестування веб-застосунків можуть відпрацьовувати нестабільно (падати на перший погляд із незрозумілих причин). Насправді це пов’язано з тим, що дані тест-кейси імітують роботу з браузером і якісь елементи можуть просто не встигнути завантажитися чи відобразитися на сторінці, оскільки час відгуку сайту може відрізнятися. Хоча він майже непомітний для людини, але тести можуть сканувати сторінку швидше ніж зміни відобразяться у вікні браузера. Саме тому під час налагодження потрібно запускати тест-кейси по декілька разів, щоб впевнитися, що в них присутня достатня кількість очікувань на відображення чи зміни елементів і вони не почнуть падати із-за вищеописаної причини.

В інтегрованому середовищі розробки IntelliJ IDEA для цього можна налаштувати конфігурацію так, щоб тести повторювалися вказану кількість разів (рисунок 20).

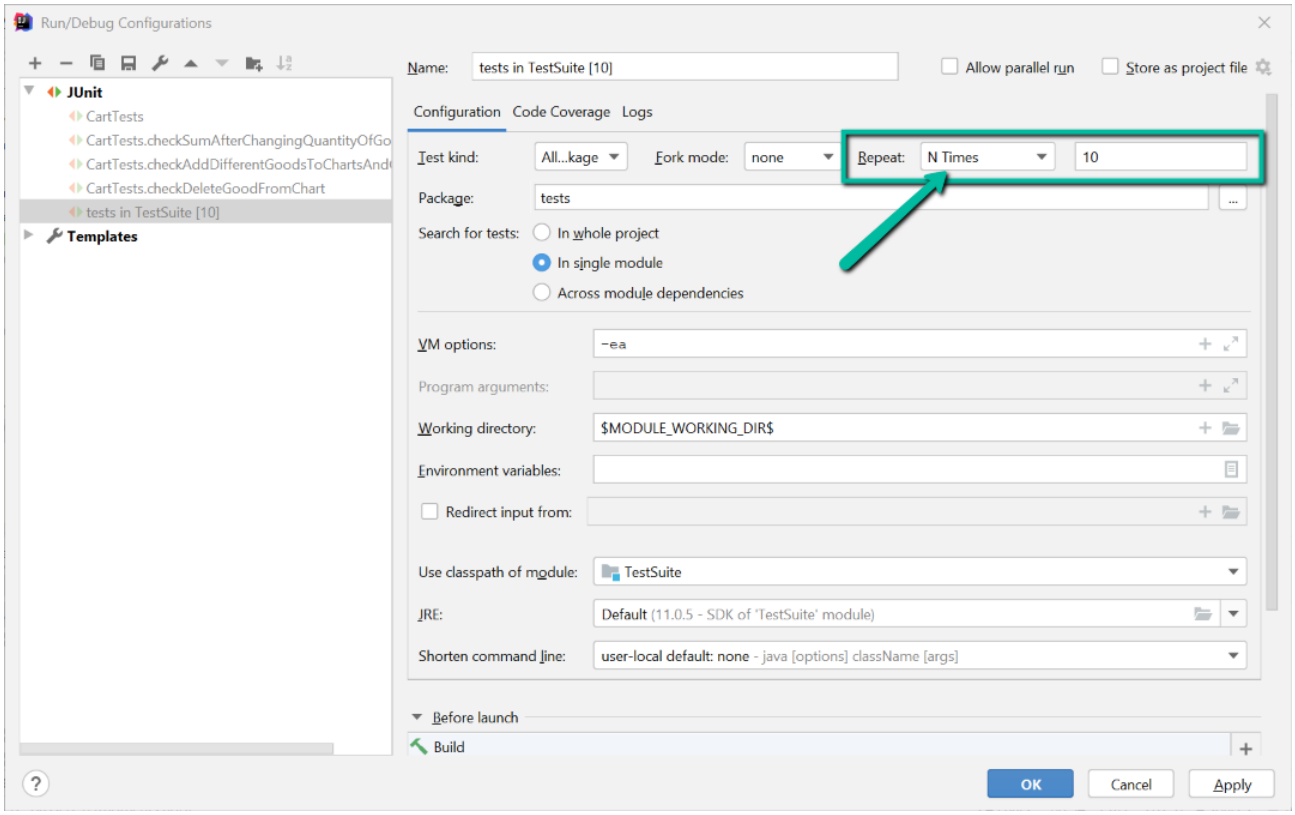

Рисунок 20 – Налаштування конфігурації

Запустимо наш набір тест кейсів 7 разів (рисунок 21).

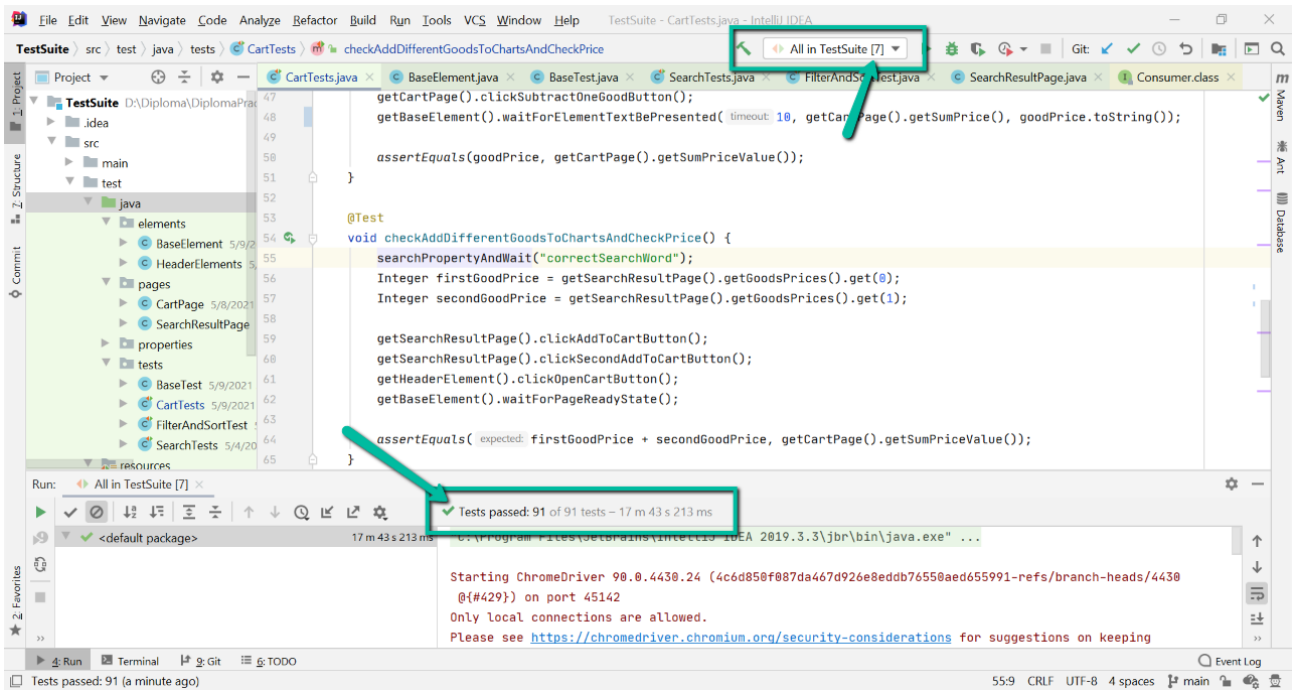

Рисунок 21 – результати запуску набору тест-кейсів 7 разів

Як можна побачити із результатів, після багаторазового запуск тест-кейсів, було отримано позитивний результат. Це свідчить про те, що даний набір тест-кейсів є стабільним і всі очікування проставлені належним чином.

Також варто звернути увагу на час виконання тест-кейсів: це менше 18 хвилин на семиразовий повтор набору тест-кейсів. З цього випливає, що на одне проходження даного набору тест-кейсів потрібно дві з половиною хвилини. Для порівняння, людина буде виконувати даний набір тест-кейсів як мінімум витрачаючи на це в декілька разів більше часу. Крім того, в такому випадку, не можна буде гарантувати стовідсоткову якість, бо не можна не враховувати присутність людського фактору.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було:

– проведено дослідження областей застосування та технологій автоматизації програмного забезпечення;

– виконано загальний огляд засобів, що застосовуються при автоматизації тестування, проаналізовано переваги та недоліки використання кожного із них;

– на базі аналізу виконаного у рамках попереднього пункту було обрано засіб для автоматизації дій веб-браузера;

– виконано дослідження та аналіз найкращих практик проєктування автоматизованих наборів тест-кейсів;

– розроблено систему автоматизованого тестування з використанням шаблонів проєктування наборів автоматизованих тест-кейсів.

При розробці системи була використана еволюційна модель. В якості інструментарію було обрано комерційне інтегроване середовище розробки IntelliJ IDEA Ultimate Edition (безкоштовно поширюване для студентів), мова програмування Java та інструмент для автоматизації дій веб-браузера Selenium WebDriver.

На сьогоднішній день автоматизація тестування веб-застосунків є невід’ємною частиною більшості великих проєктів, що швидко розвиваються. Вона стає просто необхідною при використанні гнучких ітеративних методологій розробки програмного забезпечення, оскільки з кожною новою версією виникає необхідність проводити регресійне тестування, що займає багато людського часу і не є ефективним при мануальному виконанні, тому такого роду завдання покривають автоматизованими наборами тест-кейсів.

Система автоматизації тестування інтернет-магазину може застосовуватися студентами під час опанування курсу «Програмна інженерія» та

в майбутньому навчальної дисципліни «Тестування програмного забезпечення» як зразок набору тест-кейсів для автоматизації тестування веб-застосунку. Надалі можливе доопрацювання даної системи і її розширення за рахунок покриття більшої кількості функціональності інтернет-магазину, а також впровадження нових підходів до тестування, наприклад, таких як тестування під управлінням даними.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Святослав Куликов. Тестирование программного обеспечения. Базовый курс. − 2-е издание − EPAM Systems, Версия книги 2.1.3 от 21.02.2020. − 298с. https://codernet.ru/books/qa/testirovanie_programmnogo_obespecheniya_svyatoslav_kulikov/
2. Основний сайт проекту Selenium [Електронний ресурс] – Режим доступу: https://www.selenium.dev/about/
3. Офіційний сайт Unified Functional Testing [Електронний ресурс] – Режим доступу: https://www.microfocus.com/en-us/products/uft-one/overview
4. Офіційний сайт Katalon Studio [Електронний ресурс] – Режим доступу: https://www.katalon.com/katalon-studio/
5. Офіційний сайт Watir [Електронний ресурс] – Режим доступу: http://watir.com/
6. Офіційний сайт TestComplete [Електронний ресурс] – Режим доступу: https://smartbear.com/product/testcomplete/overview/
7. TestMatick. Specific features of working with web elements while test automation. 13 January 2020 [Електронний ресурс] – Режим доступу: https://testmatick.com/specific-features-of-working-with-web-elements-while-test-automation/
8. Test Guild By Joe Colantonio. 4 Top Automation Testing Design Patterns (Plus 86 More). 13 December 2018 [Електронний ресурс] – Режим доступу: https://testguild.com/automation-testing-design-patterns/
9. GitBook. Selenium Webdriver. Уровни абстракции. Создание кастомных элементов. [Електронний ресурс] – Режим доступу: https://kreisfahrer.gitbooks.io/selenium-

webdriver/content/page_object_pattern_arhitektura_testovogo_proekta/urovni_abstraktsii_sozdanie_kastomnih_elementov.html

10. Medium. Screenplay Pattern. 11 June 2020 [Електронний ресурс] – Режим доступу: https://medium.com/testvagrant/screenplay-pattern-3490c7f0c23c
11. Хабр. Паттерны проектирования в автоматизации тестирования. 28 сентября 2017 [Електронний ресурс] – Режим доступу: https://habr.com/ru/company/jugru/blog/338836/
12. Satya Avasarala. Selenium WebDriver Practical Guide. Packt Publichng 2014. − 264c. http://padabum.com/d.php?id=173763
13. Офіційний сайт Oracle Java [Електронний ресурс] – Режим доступу: https://www.oracle.com/java/
14. Офіційний сайт Apache Maven [Електронний ресурс] – Режим доступу: https://maven.apache.org/
15. Baeldung. A Guide to JUnit 5. 26 April 2020 [Електронний ресурс] – Режим доступу: https://www.baeldung.com/junit-5
16. Хабр. Пишем автотест с использованием Selenium Webdriver, Java 8 и паттерна Page Object. 16 мая 2020 [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/502292/
17. Dima Kovalenko. Selenium Design Patterns and Best Practices. Packt Publishing 2014. − 429c. https://oiipdf.com/download/295