Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська дипломна робота на тему:

ПРОГРАМНИЙ МОДУЛЬ ТЕСТУВАННЯ WEB-ЗАСТОСУНКІВ

Виконала: студентка 4 курсу, групи 1КН-20б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Мельничук А. М.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. КН
Крилик Л. В.
(прізвище та ініціали)

« 07 » червня 2024 р.

Рецензент: к.т.н., доцент каф. АІІТ
Богач І. В.
(прізвище та ініціали)
« 07 » червня 2024 р.

Допущено до захисту
Зав. кафедри Яровий А. А.
« 07 » червня 2024 р.

Вінниця ВНТУ - 2024 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
« 07 » 03. 2024 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТЦІ
Мельничук Аміні Михайлівні

1. Тема роботи: Програмний модуль тестування WEB-застосунків
керівник роботи: Крилик Людмила Вікторівна, к.т.н., доц.
затверджені наказом вищого навчального закладу « 11 » 03. 2024 року № 80

2. Термін подання студентом роботи 27.05.2024 р.

3. Вихідні дані до роботи:
Мова програмування: об'єктно-орієнтована;
Кількість елементів тестування не більше 100 шт;
Кількість тестових сценаріїв не більше 20 од.;
Підтримка не менше трьох WEB-браузерів.

4. Зміст текстової частини (перелік питань, які потрібно розробити):
вступ; обґрунтування доцільності розробки програмного модуля тестування WEB-застосунків; розробка програмного модуля тестування WEB-застосунків; програмна реалізація модуля тестування WEB-застосунків; висновки; список використаних джерел; додатки.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень): фази моделі Водоспад; фази V-моделі; фази моделі Agile; рівні та типи тестування; розроблений чек-лист з відповідними статусами; приклади розроблених тест-кейсів; життєвий цикл дефекту; приклади розроблених баг-репортів; створені автоматизовані тести в Selenium IDE; схема алгоритму перевірки введення початкових даних; приклад роботи програми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	Завдання прийняв
1-3	Крилик Л. В., к.т.н., доц. каф. КН		

7. Дата видачі завдання 07.03.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Приміт ка
		початок	закінчення	
1	Вступ. Обгрунтування доцільності розробки програмного модуля тестування WEB-застосунків	06.05.2024	09.05.2024	
2	Розробка програмного модуля тестування WEB-застосунків	10.05.2024	14.05.2024	
3	Програмна реалізація модуля тестування WEB-застосунків	15.05.2024	21.05.2024	
4	Висновки та аналіз отриманих результатів	22.05.2024	24.05.2024	
5	Оформлення додатків	27.05.2024	29.05.2024	
6	Розробка інструкції користувача	30.05.2024	03.06.2024	
7	Оформлення матеріалів до захисту БДР	04.06.2024	06.06.2024	

Студентка _____ **Мельничук А. М**
(підпис) (прізвище та ініціали

Керівник роботи _____ **Крилик Л. В**
підпис) (прізвище та ініціали

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 123 сторінок формату А4, які включають 27 рисунків і 6 таблиць. У списку використаних джерел зазначено 27 найменувань.

Метою роботи є збільшення якості роботи програмного забезпечення автоматизованого тестування WEB-застосунків.

У загальній частині роботи на основі аналізу методів та підходів тестування доведено доцільність написання тестових сценарії, чек-листів, тест-кейсів, баг-репортів та скриптів для автоматизації тестування. В роботі розроблено алгоритм тестування обраного функціоналу WEB-застосунку.

Для розробки автоматичних тестів на основі аналізу була обрана платформа Selenium WebDriver та мова програмування Python. Для кращої демонстрації ефективності автотестування були розроблені автоматичні скрипти.

Розробка продукту була виконана в середовищі PyCharm з використанням мови програмування Python.

Ключові слова: тестування, якість програмного забезпечення, ручне тестування, автоматизоване тестування.

ABSTRACT

Bachelor's thesis consists of 123 pages of A4 format, which include 27 pictures and 6 tables. There are 27 names in the list of used sources.

The purpose of the work is to increase the quality of software for automated testing of WEB applications.

In the general part of the work, based on the analysis of testing methods and approaches, the expediency of writing test scenarios, checklists, test cases, bug reports, and scripts for testing automation has been proven. The work developed an algorithm for testing the selected functionality of the WEB application.

The Selenium WebDriver platform and the Python programming language were chosen for the development of automatic tests based on the analysis. To better demonstrate the effectiveness of self-testing, automatic scripts were developed.

Product development was performed in the PyCharm environment using the Python programming language.

Keywords: testing, quality of software, manual testing, automated testing.

ЗМІСТ

ВСТУП

Актуальність теми.

Нині сучасні технології з кожним роком не тільки роблять WEB-застосунки більш доступними та функціональними, але й ускладнюють їхнє тестування через різноманітність платформ, пристроїв та технологічні стандарти.

Мануальне тестування та модулі автоматизованого тестування є важливими етапами у процесі розробки та підтримки будь-яких ресурсів. На різних етапах тестування програмного продукту доцільно використовувати відповідні методи та підходи до тестування. Створення автоматизованих тестів не може замінити повністю роботу мануального тестувальника, адже, наприклад, іноді технології не можуть перевірити наскільки користувачу буде комфортно користуватись інтерфейсом WEB-застосунку (usability testing, UI testing). Враховуючи вимоги до функціональності, технічних обмежень та визначення оптимальних стратегій, такі підходи та методи тестування допомагають розробникам створювати інструменти, які дозволяють ефективно та якісно тестувати WEB-застосунки.

Розроблений програмний модуль тестування може знайти застосування у будь-якому проекті. Його головна функція – полегшити та пришвидшити процес тестування певного функціоналу.

Тому розробка програмного модуля тестування WEB-застосунків є актуальною натепер та може знайти застосування у галузі тестування програмного забезпечення. Це сприятиме покращенню процесу створення WEB-застосунків.

Метою дослідження є збільшення якості роботи програмного забезпечення автоматизованого тестування WEB-застосунків.

Об’єктом дослідження є процес автоматизованого тестування WEB-застосунків**.**

Предметом дослідження є програмні засоби автоматизованого тестування WEB-застосунків.

Задачі дослідження:

- обґрунтувати доцільність розробки програмного модуля тестування WEB-застосунків;

- розробити програмний модуль тестування WEB-застосунків;

- програмно реалізувати модуль тестування WEB-застосунків;

- виконати тестування програми та провести аналіз отриманих результатів

- розробити інструкцію користувача.

Апробація роботи.

Результати досліджень було апробовано на Міжнародній науково-практичній інтернет-конференції студентів аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи (МН-2024) м. Вінниці у 2024 р. [1].

Публікації: за основними результатами досліджень опубліковано тези доповіді «Аналіз передумов розробки програмного модуля тестування ВЕБ-застосунків» на науково-технічній конференції [1].

1 ОБҐРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБКИ ПРОГРАМНОГО МОДУЛЯ ТЕСТУВАННЯ WEB-ЗАСТОСУНКІВ

1.1 Основні поняття та визначення

Перш за все, потрібно ознайомитись, що ж означає тестування програмного забезпечення, що таке життєвий цикл розробки програмного забезпечення (Software Development Lifecycle (SDLC)) та життєвий цикл тестування (Software Testing Lifecycle (STLC)).

Тестування програмного забезпечення − це процес перевірки програмного продукту з метою виявлення відповідності розробки продукту до заявлених вимог. Цей процес охоплює запуск програми з різними вхідними даними та умовами, а також аналіз реакції програми на ці дані [1].

Життєвий цикл розробки програмного забезпечення включає в себе такі етапи [2]:

1. Планування та аналіз;
2. Написання вимог;
3. Дизайн архітектури продукту;
4. Розробка продукту;
5. Тестування продукту;
6. Розгортання продукту на ринку;
7. Супроводження.

Аналіз вимог є найважливішим і фундаментальним етапом в циклі життя програмного забезпечення. Він проводиться старшими членами команди за участю замовника, фахівців з продажу, маркетингових і галузевих експертів. Згодом отримана інформація використовується для планування основного підходу до проекту та проведення техніко-економічного обґрунтування продукту в економічній, операційній та технічній сферах. Планування вимог до забезпечення якості та визначення ризиків, пов'язаних з проектом, також є частиною етапу планування.

Після завершення аналізу вимог наступним кроком є створення та документування вимог до продукту та їх затвердження замовником або аналітиками ринку. Для цього створюється спеціальний документ Software Requirement Specification, (надалі SRS), тобто «Специфікація вимог до програмного забезпечення», який містить усі вимоги до продукту, що мають бути спроектовані та розроблені протягом життєвого циклу проекту.

SRS є орієнтиром для архітекторів продукту, щоб розробити найкращий дизайн архітектури продукту. На основі вимог, зазначених в SRS, зазвичай створюється декілька підходів до проектування архітектури продукту, які документуються в DDS - специфікації проектної документації (Design Document Specification). DDS розглядається всіма ключовими зацікавленими сторонами і на основі різних параметрів, таких як оцінка ризиків, надійність продукту, модульність дизайну, бюджетні та часові обмеження, вибирається найкращий підхід до проектування продукту. На етапі дизайну архітектури продукту чітко визначають всі архітектурні модулі продукту, а також їхню комунікацію та представлення потоків даних із зовнішніми та сторонніми модулями (якщо такі є). Внутрішній дизайн всіх модулів запропонованої архітектури має бути чітко визначений в DDS до найдрібніших деталей.

На наступному етапі циклу життя програмного забезпечення починається власне розробка та створення продукту. На цьому етапі створюється програмний код відповідно до DDS.

На етапі тестування продукту, відповідно до обраного підходу розробки програмного забезпечення, тестування продукту може бути як окремим етапом так і підмножиною всіх етапів, оскільки в сучасних моделях циклу життя програмного забезпечення тестування, як правило, відбувається на всіх етапах. Однак цей етап стосується лише тестування продукту, на якому фіксуються дефекти продукту, відстежуються, виправляються і повторно тестуються, поки продукт не буде відповідати стандартам якості, визначеним в SRS.

Після того, як продукт пройшов тестування і готовий до впровадження, його офіційно випускають на відповідний ринок. Потім, на основі зворотного зв'язку,

продукт може бути випущений як є або із запропонованими покращеннями для цільового сегменту ринку.

Після того, як продукт випущено на ринок, його підтримують для існуючої клієнтської бази.

Існують різні моделі життєвого циклу розробки програмного забезпечення (SDLC). Кожна модель процесу слідує набору кроків, унікальних для свого типу, щоб забезпечити успіх у процесі розробки програмного забезпечення. Найбільш використовуваними моделями є «Водоспад» (Waterfall Model), V-модель, модель великого вибуху (Big Bang Model), Agile модель та інші.

Waterfall була першою моделлю SDLC, яку почали широко використовувати в розробці програмного забезпечення для забезпечення успіху проекту. Зазвичай таку модель використовують для великих проектів з високим рівнем вимог до якості продукту, таких як космічна галузь, медицина тощо. У водоспадному підході весь процес розробки програмного забезпечення поділяється на окремі фази. У цій моделі вихід однієї фази, як правило, виступає в якості входу для наступної фази в послідовності [3]. На рисунку 1.1 показані різні фази моделі водоспаду.

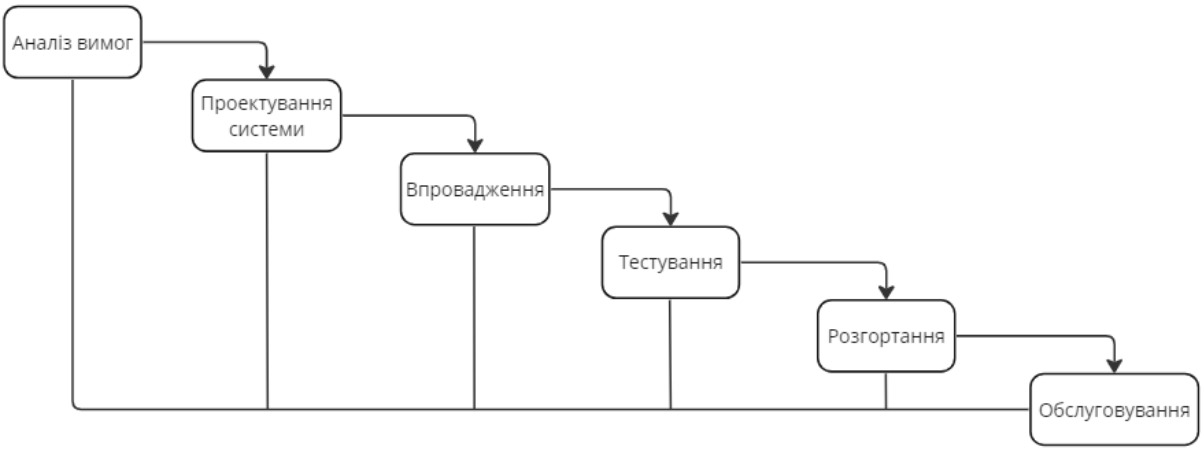


Рисунок 1.1 – Фази моделі Водоспад

Наведемо деякі з основних переваг моделі водоспаду:

- проста і легка для розуміння і використання;

- легко керувати завдяки жорсткості моделі, кожна фаза має конкретні результати і процес перевірки;
- фази обробляються і завершуються по черзі;
- добре підходить для невеликих проектів, де вимоги дуже добре зрозумілі;
- чітко визначені етапи;
- добре зрозумілі проміжні результати;
- легко організувати завдання;
- процес і результати добре задокументовані.

Основними недоліками моделі водоспаду є такі:

- робоче програмне забезпечення не створюється до кінця життєвого циклу;
- високий рівень ризику та невизначеності;
- не найкраща модель для складних та об'єктно-орієнтованих проектів;
- погана модель для тривалих і безперервних проектів;
- не підходить для проектів, де вимоги мають помірний або високий ризик зміни. Отже, ризик і невизначеність у цій моделі процесу високі;
- важко виміряти прогрес на етапах;
- не може врахувати мінливі вимоги;
- коригування обсягу робіт протягом життєвого циклу може призвести до завершення проекту.

V-модель - це модель SDLC, в якій процеси виконуються послідовно у V-подібній формі. Вона також відома як модель верифікації та валідації.

V-модель є розширенням моделі водоспаду і базується на асоціюванні фази тестування з кожною відповідною фазою розробки. Це означає, що для кожної окремої фази циклу розробки існує безпосередньо пов'язана з нею фаза тестування. Це дуже дисциплінована модель, і наступна фаза починається тільки після завершення попередньої фази [4]. На рисунку 1.2 зображені фази V-моделі.

Переваги методу V-моделі полягають у наступному:

- це дуже дисциплінована модель і фази виконуються по черзі;
- добре підходить для невеликих проектів, де вимоги добре зрозумілі;
- проста і легка для розуміння і застосування;

- легко керувати завдяки жорсткості моделі. Кожна фаза має конкретні результати і процес перевірки.

Недоліки методу V-моделі:

- високий ризик і невизначеність;
- не підходить для складних та об'єктно-орієнтованих проектів;
- не найкраща модель для тривалих і безперервних проектів;
- не підходить для проектів, де існує середній або високий ризик зміни вимог;
- після того, як додаток знаходиться в процесі тестування, важко повернутися назад і змінити функціональність.
- робоче програмне забезпечення не створюється до кінця життєвого циклу.

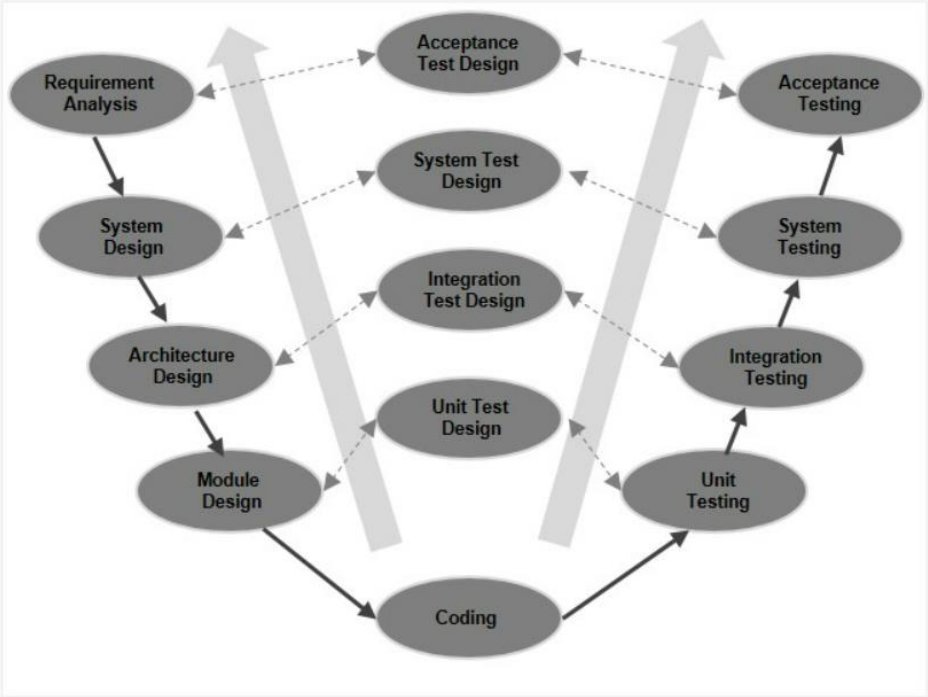


Рисунок 1.2 – Фази V-моделі

Модель Agile SDLC − це поєднання ітеративних та інкрементних моделей процесів з акцентом на адаптивність процесів та задоволення потреб клієнтів шляхом швидкої доставки робочого програмного продукту. Гнучкі методи розбивають продукт на невеликі інкрементні збірки. Ці збірки виконуються в ітераціях. Кожна ітерація зазвичай триває від одного до трьох тижнів. У кожній

ітерації беруть участь міжфункціональні команди, які одночасно працюють над різними напрямками, такими як планування, дизайн, реалізація, тестування, перегляд та впровадження [5]. Кожна ітерація називається спринтом. На рисунку 1.3 зображені фази моделі Agile.

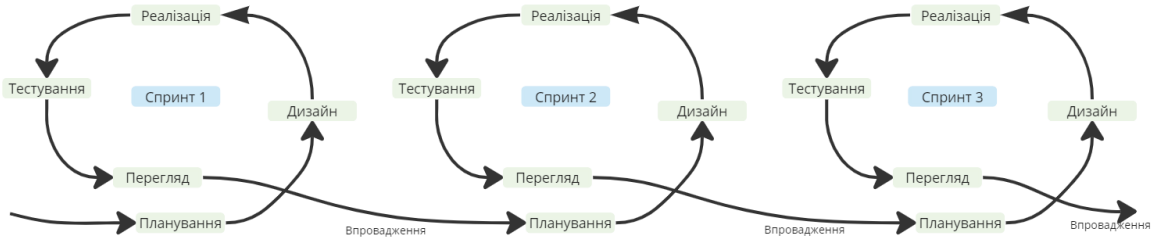


Рисунок 1.3 − Фази моделі Agile

Модель Agile включає в себе 2 підходи до управління проектом: Scrum і Kanban. В таблиці 1.1 подано основні відмінності між цими двома підходами.

Таблиця 1.1 − Основні відмінності між Scrum і Kanban

Scrum	**Kanban**
Щоденні зустрічі (мітинги)	Щоденні зустрічі не обов’язкові
Ретроспектива (мітинг у кінці кожного спринта, на якому обговорюють як пройшов спринт, що потрібно змінити для кращої роботи в наступному спринті і т.д)	Не обов’язковою є ретроспектива
Є ролі Scrum-майстер та Product owner	Присутня лише роль Team-лід
Є оцінювання кожного завдання.	Оцінка завдань є необов’язковою.
Пріоритети під час спринта не можуть змінюватись.	Пріоритети можна змінювати будь-коли.
В кожному спринті має бути виконана визначена наперед кількість конкретних завдань	Кожен член команди розробки обирає тільки один тікет (завдання) з беклогу на виконання. Наступний тікет може бути обраний лише після закінчення попереднього.

1.2 Аналіз підходів до тестування програмного забезпечення

1.2.1 Проактивне і реактивне тестування

У таблиці 1.2 представлена основна різниця між проактивним та реактивним підходами тестування.

Таблиця 1.2 – Відмінність між проактивним та реактивним підходами тестування

Проактивне	Реактивне
Цей підхід до тестування, у якому процес тестування розпочинається якомога раніше, з метою знаходження та виправлення дефектів до створення збірки	Цей підхід, у якому тестування починається після завершення кодування
Цей підхід передбачає розглядати наперед нові можливі ситуації, а не просто працювати за планом	Реактивний підхід передбачає реагувати на проблеми, коли вони з'являються, а не запобігати їх утворенню

1.2.2 Мануальне і автоматизоване тестування

На сьогодні все частіше на проектах можна зустріти як мануальних, тобто ручних тестувальників, так і автоматизаторів. Обидві професії мають свої переваги та недоліки та не можуть повністю замінити один одного.

Ручні тестувальники можуть працювати, шукаючи нові помилки, які не були попередньо передбачені. Вони можуть використовувати різні комбінації вхідних даних, використовувати нестандартні сценарії та експериментувати з функціональністю програми. Однією із важливих переваг, яка передбачає взаємодію з користувачем та здатність враховувати вподобання та вимоги реальних користувачів є те, що тестувальники можуть оцінити, наскільки програмне забезпечення задовольняє потреби користувачів та виявити проблеми, пов'язані з інтерфейсом або функціоналом, які впливають на задоволеність користувачів. В той час як автоматизовані тести працюють з блискавичною швидкістю, набагато оперативніше, ніж люди. Вони можуть виконувати повторювані завдання з дивовижною швидкістю, що дозволяє економити

безцінний робочий час розробників. Автоматизовані тести завжди стабільні та надійні. Немає небажаних «людських помилок» чи недоліків, пов’язаних з втомою чи недостатньою концентрацією. Кожен тест повторюється точно так само кожного разу, забезпечуючи надійну перевірку функціонала [6].

Автоматизоване тестування має такі переваги:

- працюють з блискавичною швидкістю;
- завжди стабільні та надійні;
- зменшення часу на виявлення та виправлення помилок;
- регресійне тестування легше реалізується завдяки автоматизації.

До недоліків автоматизованого тестування можна віднести значні початкові витрати, неможливість абсолютної заміни ручного тестування, наявність досвідченого персоналу та потребу постійного оновлення.

Мануальне (ручне) тестування має такі переваги:

- гнучкість, яка дозволяє тестувальникам досліджувати різні аспекти програмного забезпечення, розігрувати різні сценарії та швидко адаптуватися до мінливих умов;
- здатність експериментувати та реакція на непередбачуваність;
- інтуїтивність;
- швидкість навчання та простота освоєння;
- креативність та можливість виявити нові проблеми;
- адаптивність до змін.

До недоліків ручного тестування можна віднести:

- людський фактор;
- витрати;
- ручне тестування складно масштабувати для великих проєктів або продуктів зі складною функціональністю;
- ручне тестування може бути обмеженим у виявленні деяких проблем, особливо тих, які пов’язані з великим обсягом даних або високою навантаженістю системи.

1.2.3 Білий, чорний, сірий ящик

Техніка тестування без знання внутрішньої роботи програми називається тестуванням "чорного ящика". Тестувальник не знає архітектури системи і не має доступу до вихідного коду. Зазвичай під час тестування "чорного ящика" тестувальник взаємодіє з користувацьким інтерфейсом системи, надаючи вхідні дані та перевіряючи вихідні, не знаючи, як і де обробляються вхідні дані [7].

Переваги тестування "чорного ящика":

- Добре підходить і ефективний для великих сегментів коду;
- Доступ до коду не потрібен;
- Чітко відокремлює точку зору користувача від точки зору розробника завдяки чітко визначеним ролям;
- Велика кількість тестувальників може протестувати додаток, не маючи знань про реалізацію, мову програмування чи операційну систему.

Недоліки даного підходу можуть бути такими:

- Обмежене покриття, оскільки фактично виконується лише вибрана кількість тестових сценаріїв;
- Неефективне тестування через те, що тестувальник має обмежені знання про додаток;
- Сліпе покриття, оскільки тестувальник не може націлитися на конкретні сегменти коду або області, схильні до помилок;
- Тестові кейси складно розробляти.

Тестування "білого ящика" - це детальне дослідження внутрішньої логіки та структури коду. Для того, щоб виконати тестування "білого ящика" на додатку, тестувальник має знати внутрішню роботу коду.

Тестувальник має зазирнути у вихідний код і з'ясувати, який блок/частина коду поводиться неналежним чином.

Переваги тестування "білого ящика":

- Оскільки тестувальник знає вихідний код, стає дуже легко з'ясувати, який тип даних може допомогти в ефективному тестуванні програми;
- Це допомагає оптимізувати код;

- Можна видалити зайві рядки коду, які можуть містити приховані дефекти;

- Завдяки знанням тестувальника про код досягається максимальне покриття під час написання тестового сценарію.

Недоліками тестування "білого ящика" є:

- Збільшення витрат на кваліфікованого спеціаліста;

- Іноді неможливо зазирнути в кожен куточок, щоб виявити приховані помилки, які можуть створити проблеми, оскільки багато шляхів залишаться неперевіреними;

- Такий підхід є складним в реалізації, оскільки він вимагає спеціалізованих інструментів.

Тестування "сірого ящика" - це метод тестування програми з обмеженими знаннями про внутрішню роботу програми. Знання предметної області системи завжди дає тестувальнику перевагу над тими, хто має обмежені знання в цій області. На відміну від тестування "чорного ящика", де тестувальник тестує лише користувацький інтерфейс програми, при тестуванні "сірого ящика" тестувальник має доступ до проектної документації та бази даних. Маючи ці знання, тестувальник може підготувати кращі тестові дані та тестові сценарії під час складання плану тестування [8].

Переваги такого підходу:

- Тестувальники сірого ящика не покладаються на вихідний код; натомість вони покладаються на визначення інтерфейсу та функціональні специфікації;

- На основі обмеженої доступної інформації тестувальник сірого ящика може розробити чудові тестові сценарії, особливо для протоколів зв'язку та обробки типів даних;

- Тестування проводиться з точки зору користувача, а не дизайнера.

Недоліки:

- Оскільки доступ до вихідного коду недоступний, можливість переглядати код і тестове покриття обмежена;

- Тести можуть бути надлишковими, якщо розробник програмного забезпечення вже запустив тестовий кейс;
- Тестування кожного можливого вхідного потоку є нереальним, оскільки це займе невиправдано багато часу; отже, багато шляхів програми залишаться неперевіреними.

1.2.4 Верифікація і валідація

Верифікація – це процес перевірки вимог, програмного коду, документації, тест кейсів тощо. Зазвичай на етапі верифікації тестувальники часто звертаються до статичного підходу тестування.

Валідація – це безпосередньо перевірка роботи програми. На цьому етапі часто застосовується динамічний підхід тестування. У таблиці 1.3 представлено відмінності між верифікацією та валідацію [9].

Таблиця 1.3 – Різниця між верифікацією та валідацією

Верифікація	**Валідація**
Процес верифікації включає перевірку документів, дизайну, коду та програми	Це динамічний механізм тестування та валідації фактичного продукту
Він не передбачає виконання коду Верифікація використовує такі методи, як огляди, обстеження, інспекції, камеральна перевірка тощо.	Це завжди пов'язано з виконанням коду
Верифікація використовує такі методи, як огляди, обстеження, інспекції, камеральна перевірка тощо.	Він використовує такі методи, як тестування "чорного ящика", тестування "білого ящика" та нефункціональне тестування
Перевіряється, чи відповідає програмне забезпечення специфікації	Він перевіряє, чи відповідає програмне забезпечення вимогам та очікуванням замовника
Знаходить помилки на ранніх стадіях циклу розробки	Він може знаходити помилки, які процес верифікації не може зловити

Продовження таблиці 1.3

Верифікація	Валідація
Команда QA виконує перевірку і переконується, що програмне забезпечення відповідає вимогам, викладеним у документі SRS.	Із залученням команди тестувальників виконується валідація програмного коду.
Відбувається перед валідацією	Відбувається після верифікації

1.3 Типи тестування

У процесі тестування існують різні рівні. Вони включають в себе різні методології, які можуть бути використані при проведенні тестування програмного забезпечення. Основними можна виділити статичні і динамічні типи тестування [8]. Динамічні типи тестування в свою чергу поділяються на функціональні, нефункціональні та типи обслуговування. На рисунку 1.4 зображена схема рівнів та типів тестування.

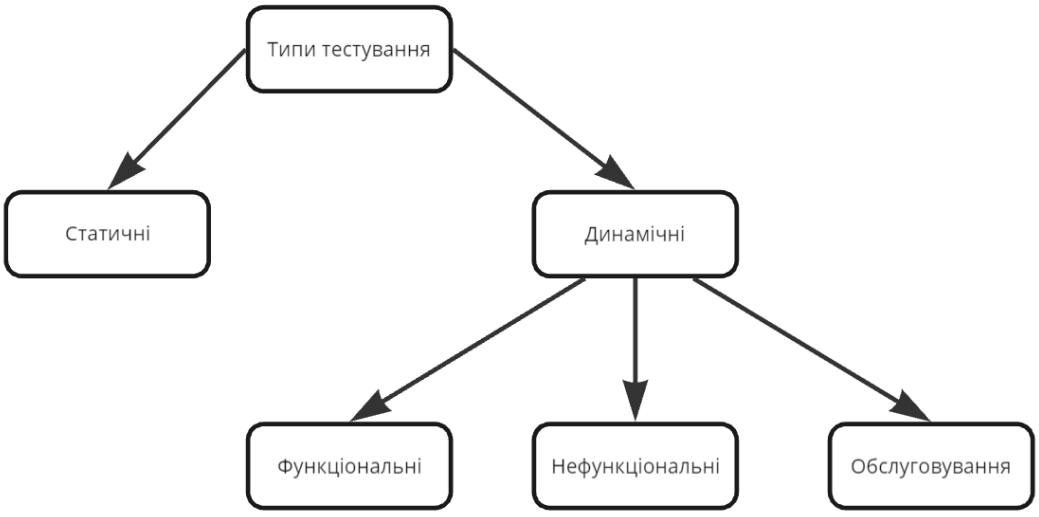


Рисунок 1.4 – Рівні та типи тестування

1.3.1 Функціональні типи

Цей тип тестування "чорного ящика", який базується на специфікаціях програмного забезпечення, що тестується. Додаток тестується шляхом введення вхідних даних, а потім досліджуються результати, які мають відповідати

функціональності, для якої він був призначений. Функціональне тестування програмного забезпечення проводиться на повній, інтегрованій системі для оцінки відповідності системи заданим вимогам.

До функціональних типів тестування можна віднести:

- модульне тестування (unit);
- інтеграційне тестування (integration);
- димне тестування (smoke);
- тестування прийняття (acceptance);
- перевірка перекладу сторінки (localization);
- тестування глобалізації (globalization);
- тестування на функціональну сумісність (interoperability).

Модульне тестування зазвичай роблять безпосередньо програмісти за допомогою авто-тестів. Інтеграційне тестування включає в себе певну кількість юніт тестів. Під час димного тестування перевіряється чи після оновлення програма працює справно та виконує основні функції. Тестування прийняття зазвичай є останнім етапом в тестуванні продукту. Під час нього клієнт перевіряє чи продукт відповідає усім вимогам та чи може він бути прийнятим до використання.

1.3.2 Нефункціональні типи

Цей розділ базується на тестуванні програми з точки зору її нефункціональних атрибутів. Нефункціональне тестування передбачає перевірку програмного забезпечення на відповідність вимогам, які є нефункціональними за своєю природою, але важливими, такими як продуктивність, безпека, користувацький інтерфейс і т.д.

Деякі з важливих і часто використовуваних типів нефункціонального тестування:

- тестування продуктивності (performance);
- навантажувальне тестування (load);
- тестування на витривалість (endurance);

- тестування зручності (usability);
- тестування безпеки (security).

Тестування продуктивності виконується для того, щоб розуміти наскільки швидко та коректно працює програма під певним навантаженням. Навантажувальне тестування проводиться для розуміння поведінки програми під певним очікуваним навантаженням, а тестування на витривалість перевіряє роботу програми за межами нормальної експлуатаційної спроможності. Тестування зручності полягає в тому, щоб зрозуміти наскільки зручно та комфортно буде користувачу взаємодіяти з програмою чи сайтом. Тестування безпеки проводиться для перевірки безпеки системи та захисту від атак хакерів.

1.3.3 Типи обслуговування

До типів обслуговування (maintenance) можна віднести такі типи: регресивне тестування (regression) та тестування обслуговування (maintenance). Регресивне тестування направлене на перевірку змін, зроблених з кожною новою версією. Тестування обслуговування − це випробування, яке проводиться з метою виявлення проблем з обладнанням, діагностики проблем з обладнанням або підтвердження того, що ремонтні заходи були ефективними.

1.4 Висновок

В цьому розділі було проведено аналіз основних понятть та визначень в області тестування програмного забезпечення (ПЗ). Для проведення ефективного тестування потрібно вцілому володіти інформацією про сам процес розробки ПЗ. Тому першочергово уваза була приділена дослідженню таких понять як методи та процеси розробки ПЗ. Що саме являє собою життєвий цикл розробки програмного забезпечення (SDLC) та життєвий цикл тестування (STLC). Окреслено та достіджено документацію, яка використовується при розробці та основні тестові артефакти. Їх роль, важливість, сфера використання.Також, окрема увага була приділення виченню понятть тестові підходи, типи, інструменти. Це дає нам

розуміння, які краще методи та підходи використовувати при написанні певних тестових документів, такі як тест план, тест сценарії, тест кейси.

Тема налаштування ефективного процесу тестування є актуальною в наш час, оскільки рівень вимог клієнтів та користувачів до розроблених продуктів зростає. Тестування дає змогу оцінити відповідність розробки певного продукту до вимог замовника. Це допомагає зробити певні висновки та спланувати подальщі дії команди для покращення продукту.

2 РОЗРОБКА ПРОГРАМНОГО МОДУЛЯ ТЕСТУВАННЯ WEB-ЗАСТОСУНКІВ

Тестові артефакти – це набір документів та скриптів, які створюються протягом життєвого циклу тестування програмного забезпечення (STLC), щоб гарантувати, що протестований додаток відповідає бажаним стандартам якості. Ці артефакти надають докази процесу тестування та цінну інформацію про загальні зусилля з тестування.

Існують різні типи тестових артефактів, кожен з яких служить певній меті. Тестові плани, тестові кейси та тестові скрипти - одні з найпоширеніших тестових артефактів, що використовуються при тестуванні програмного забезпечення. Розуміння різних типів тестових артефактів та їх призначення є важливим для ефективного тестування програмного забезпечення.

На потребу в створенні артефактів тестування можуть впливати декілька факторів:

1. Специфіка проекту: галузі, що потребують найвищого рівня контролю виконання та якості, наприклад, аерокосмічна, ядерна енергетика чи охорона здоров'я;

2. Потреби зацікавлених сторін: команда тестувальників, розробники, менеджери проектів і замовники можуть мати різні уявлення про те, що їм потрібно від результатів тестування. Розробникам можуть знадобитися детальні звіти, щоб зрозуміти і вирішити проблеми, в той час як клієнтам може знадобитися резюме про те, наскільки якісним і надійним є продукт;

3. Мета тестування також відіграє важливу роль у визначенні того, які результати тестування та в якій формі можуть допомогти в оцінці стадії розробки продукту. Більш детальні та обширні результати тестування можуть допомогти знайти і виправити якомога більше проблем до релізу. Але якщо мета - просто забезпечити базовий рівень якості, більш детальні результати тестування можуть бути непотрібні.

Типи тестових артефактів:

- Тестова стратегія;
- Тест план;
- Тестовий сценарій;
- Тест кейси;
- Матриця покриття;
- Звіт тестування.

Стратегія тестування зазвичай готується менеджером тестування або менеджером проекту на рівні керівництва. Це план документа, який описує підхід до тестування в циклі розробки програмного забезпечення, в якому перераховано, як досягти очікуваного результату, використовуючи наявні ресурси [10].

Тестова стратегія забезпечує легке розуміння цілей, інструментів, методів, інфраструктури та термінів проведення тестових заходів, які повинні бути виконані. Також використовується для визначення всіх факторів ризику, які можуть виникнути під час тестування, і відповідних рішень для зменшення або пом'якшення ризиків. Тест стратегія також прояснює основні важливі виклики та підхід до завершення всього процесу тестування проекту. Стратегія тестування зазвичай випливає з формату специфікації бізнес-вимог.

Тестова стратегія може включати опис таких моментів [11]:

- основна мета тестування;
- керівні принципи проведення тестування;
- вказати вимоги, необхідні для проведення тестування, такі як функціональні вимоги, тестові сценарії, ресурси і т. д.;
- функціонал, що буде підлягати тестуванню та функціонал, що не буде включений в перелік тестування;
- ролі та обов'язки членів команди тестування;
- необхідні рівні тестування;
- техніки тест дизайну, що будуть використовуватись під час тестування;
- основний очікуваний результат тестування;
- ризики;
- методи попередження виникнення проблем.

2.1 Користувацькі сценарії для тестування веб-застосунків

Користувацький сценарій тестування − це коротке, стисле пояснення продукту або його можливостей з точки зору кінцевого користувача. У Agile − це невелика одиниця робочого процесу. Вона дозволяє команді розробників зрозуміти бажання і потреби користувача і гарантує, що програма відповідає цим вимогам. Користувацькі сценарії часто використовуються також при розробці програмного забезпечення [12].

Такий артефакт має важливе значення для керування процесом розробки та забезпечення відповідності кінцевого продукту вимогам споживача. Як користувач, я прагну мати функціонал, щоб мета була структурованою. Наприклад, я хочу відстежувати статус своєї покупки онлайн і знати, коли очікувати доставку. Тестувальники мають розуміти історії користувачів і перетворювати їх на привабливі тестові кейси, щоб переконатися, що програма функціонує належним чином.

Сценарій користувача − гарний компонент для будь-якої команди розробників, але є кілька важливих характеристик, які потрібно враховувати при їх створенні, а саме:

- Швидке та стисле пояснення функції або потреби програмного забезпечення;
- Написане з точки зору користувача або зацікавленої сторони;
- Критерії прийнятності для визначення очікуваної поведінки функції. Може бути розділений на менші, більш керовані частини (епізоди, теми або завдання);
- Часто написаний у форматі "користувач, функціональність, мета", команда розробників і зацікавлені сторони досягають спільного розуміння;
- Пріоритети визначаються на основі бізнес-цінності та впливу на досвід кінцевого користувача.

Користувацькі сценарії, як правило, є одиницею роботи або завданням в рамках гнучкого фреймворку, і вони допомагають команді розробників зрозуміти цінність того, що вони розробляють, з нетехнічної точки зору.

Користувацькі сценарії є важливим компонентом гнучкого фреймворку (Agile), оскільки вони забезпечують підхід, орієнтований на людину до повсякденної роботи розробників. Сценарії користувачів полегшують розробникам співпрацю та обговорення роботи з членами команди, а також визначають більш чіткий шлях до створення чогось цінного для користувача. А оскільки користувач перебуває в центрі уваги, загальна якість продукту підвищується.

В рамках гнучкого фреймворку проектні команди організовують роботу в певних структурах залежно від їх обсягу та розміру. Поширені назви цих структур включають Initiatives, Epic, User story та Substory [13]:

- Initiatives: збірка епічних творів зі спільною метою;
- Epic: більша група робіт, що включає серію користувацьких історій;
- User story: невеликий запит на функцію або досвід, написаний з точки зору користувача;
- Substory: більш детальна історія користувача, що використовується для надання додаткової інформації про конкретний досвід користувача.

Найпоширеніша структура користувацького сценарію - це речення зі змінними, які можна змінювати і які стосуються конкретного користувацького сценарію. Ось структура користувацького сценарію:

"Як [користувач/клієнт/персонаж], я [хочу], [щоб]"

- Аудиторія або "користувач/клієнт/персонаж": ця частина пояснює персону або тип клієнта, для якого буде створюватися продукт. Сегменти ринку та персони ідеально підходять для надання всебічного пояснення типу людей, на яких розрахований цей сценраії. Якщо у вас немає ні того, ні іншого, ми рекомендуємо взяти клієнта, якого знає ваша команда і який представляє аудиторію, для якої ви плануєте розробляти продукт. Це дозволить команді розробників проникнути в образ мислення користувача і бути більш емпатичними до його потреб;

- Намір або "хочу": у цьому розділі ви описуєте, чого хоче досягти користувач. Йдеться не про функцію (функції) чи інтерфейс, який вони використовують, а лише про їхні цілі. Вона має бути вільною від вашого поточного рішення, жаргону та особливостей;

- Вигода або "щоб": тут вказується, що клієнт прагне вирішити або яку вигоду він намагається отримати. Це "чому", що стоїть за тим, що потрібно вашому користувачеві від рішення.

Нижче представлено 10 розроблених користувацьких сценаріїв для функціоналу WEB-застосунку.

10 user stories for functionality:

1. As a user, I want to have the opportunity to create my own personal account.

2. As a user, I want to add the product to the wishes, so then I can buy it.

3. As a user, I want to add the product to the cart, so I can buy it immediately.

4. As a user, I want to have the opportunity to add a photo to my personal account.

5. As a user, I want to have the opportunity to add the main delivery address.

6. As an admin, I want to have access to databases, so that I can add new registered users there to store their personal information.

7. As an admin, I want to delete inappropriate comments, so all customers won't see them.

8. As an admin, I want to have the opportunity to ban the user for incorrect actions to restrict his access to the site.

9. As an admin, I want to have the opportunity to add new books to the range of the store.

10. As an admin, I want to receive notifications of messages that users send to support, so that I can respond to them.

А також 10 розроблених користувацьких сценаріїв для інтерфейсу WEB-застосунку.

10 user stories for UI:

1. As a user, I want the background of the site to be in white and gray tones.

2. As a user, I want all pages of the site to contain the bookstore logo at the upper left corner.

3. As a user, I want the appearance of the cursor to change when I hover the mouse over the link.

4. As a user, I want the home page to contain a carousel of banners.

5. As a user, I want site navigation and contacts to be placed in the footer.

6. As a user I want the page of the book to contain a photo of this book.

7. As a user, I want each page to contain breadcrumbs navigation at the beginning.

8. As a user, I want to have round logos of social networks with links at the bottom of the footer.

9. As a user, I want to see a carousel of popular authors at the home page.

10. As a user, I want to have a brief description of the bookstore before the footer on the main page.

2.2 Специфікація вимог до програмного забезпечення

Специфікація вимог до програмного забезпечення (SRS) − це повний документ або опис вимог до системи або програмного додатку. Іншими словами, "SRS − це документ, який описує, якими будуть функції програмного забезпечення і якою буде його поведінка, тобто як воно буде працювати". Перевага SRS полягає в тому, що розробникам потрібно менше часу і зусиль для розробки програмного забезпечення. SRS - це як макет програмного забезпечення, який користувач може переглянути і побачити, чи відповідає він (SRS) його потребам чи ні [14]. Ми дізналися про специфікацію вимог до програмного забезпечення, що це таке, тепер давайте ознайомимося з її особливостями

Розглянемо специфікацію вимог до програмного забезпечення:

Важливість специфікації вимог до програмного забезпечення (SRS):

1. Ясність і розуміння:

SRS забезпечує загальний канал зв'язку між зацікавленими сторонами, щоб усі погоджувалися з тим, на що спрямований проект. Це зменшує двозначність і ризики непорозумінь між членами команди при чіткому визначенні вимог.

2. Основа для планування проекту:

Планування проектів з розробки програмного забезпечення має бути особливо детальним. SRS − це необхідна інформація для менеджерів проектів для визначення реалістичних термінів, призначення відповідних ресурсів та встановлення проміжних результатів. Він забезпечує основу для планування проектів, що дозволяє командам точно оцінювати витрати та графіки.

3. Керівництво для команд розробників:

SRS використовується для того, щоб допомогти розробникам зрозуміти функціональні специфікації і те, як буде розроблятися програмне забезпечення. Документ пропонує огляд архітектури системи, користувацьких інтерфейсів та структур даних, які дозволяють розробникам створити систему, що відповідає очікуванням клієнта.

4. Сприяє тестуванню та забезпеченню якості:

SRS використовуються як посилання командами забезпечення якості та тестування, які використовують їх для розробки тестових кейсів. Перевіряють, чи відповідає програмне забезпечення тому, що було визначено. Оскільки тестування має чітко визначені вимоги, воно стає набагато більш систематичною процедурою і ймовірність того, що важлива функціональність буде пропущена, зменшується.

5. Управління змінами:

Під час проектів зміни неминучі. Таким чином, система збереження змін забезпечує базову лінію, з якої можна оцінювати будь-які запропоновані зміни. Це гарантує, що зміни ретельно реєструються, затверджуються та належним чином впроваджуються − без "повзучості" обсягу робіт.

6. Задоволеність клієнта та зацікавлених сторін:

SRS − це контракт між командою розробників та їхнім клієнтом або зацікавленими сторонами. Це підвищує задоволеність та довіру клієнта, якщо кінцевий продукт відповідає тому, що було задокументовано. Крім того, така чітка

документація забезпечує основу для вирішення суперечок або розбіжностей під час розробки.

7. Зменшення ризиків:

Чим раніше в життєвому циклі розробки програмного забезпечення ви зможете виявити потенційні ризики, тим краще. SRS допомагає в оцінці ризиків, визначаючи залежності, обмеження та можливі ризики. Це робить процес розробки більш плавним для менеджерів проектів, які потім можуть придумати стратегії для запобігання ризикам.

8. Покращена співпраця:

Розробка програмного забезпечення вимагає співпраці. SRS має командний дух. Працюючи разом з міжфункціональними командами, вона формує єдине бачення та спільне розуміння проекту. В результаті процес розробки стає більш інтегрованим та ефективним.

Особливості специфікації вимог до програмного забезпечення (SRS) [14]:

- Повнота: SRS має бути повним, тобто всі вимоги до програмного забезпечення потрібно згадати в SRS;
- Коректність: воно має бути коректним, тобто відповідати потребам замовника;
- Чіткий: він має бути чітким. Вимоги до програмного забезпечення мають бути чітко задекларованими.;
- Точний: він має мати точність. Якщо воно не є точним, то програмне забезпечення не може бути розроблене;
- Послідовний: має бути послідовним від початку до кінця, щоб користувачі могли легко зрозуміти вимоги. А узгодженість може бути досягнута тільки тоді, коли немає протиріч між двома вимогами;
- Перевірюваність: має бути верифікованими. Вимоги перевіряються експертами та тестувальниками;
- Модифікується: всі вимоги, зазначені в документі SRS, мають бути модифіковані. Це може статися лише тоді, коли структура системи документів є збалансованою;

- Відстежуваність: має бути простежуваною, тобто кожна вимога в ній має бути ідентифікована по-різному. Кожна вимога має мати власну ідентичність;
- Перевірюваність: має бути тестованою, тобто її можна перевірити будь-яким способом;
- Однозначний: може бути однозначним лише тоді, коли всі вимоги мають лише одне значення. Тобто має бути лише одна інтерпретація.

Нижче представлені 10 вимог до функціоналу WEB-застосунку, розроблені на основі користувацьких сценаріїв.

Requirements from user stories for functionality:

1. System should create personal accounts for all registered users.
2. System should allow users to add the product to their wishes, so then they can buy it.
3. System should allow users to add the product to their cart, so they can buy it immediately.
4. Users should have the opportunity to add a photo to their personal account.
5. Users should have the opportunity to add the main delivery address.
6. Admin should have access to databases.
7. Admin should have the rights to delete inappropriate comments.
8. Admin should have the opportunity to ban the user for incorrect actions.
9. Admin should have the opportunity to add new books to the range of the store.
10. System should send notifications of messages from users to the admin.

А також 10 вимог до інтерфейсу WEB-застосунку, розроблені на основі користувацьких сценаріїв.

Requirements from user stories for UI:

1. The background of the site should be in white and gray tones.
2. All pages of the site must contain the bookstore logo at the upper left corner.
3. The appearance of the cursor should change when the user hover the mouse over the link.
4. The home page should contain a carousel of banners.
5. Site navigation and contacts should be placed in the footer.

6. The page of the book should contain a photo of this book.

7. Each page should contain breadcrumbs navigation at the beginning.

8. Round logos of social networks with links should be placed at the bottom of the footer.

9. The home page should contain a carousel of popular authors.

10. Before the footer on the main page should be a brief description of the bookstore.

2.3 Чек-лист

Чек-лист є фундаментальним елементом тестування програмного забезпечення. Він включає в себе ряд тестів, які допомагають визначити, чи готовий продукт до розгортання. А якщо ні, то допомагає з'ясувати, які компоненти необхідно доопрацювати.

Чек-лист тестування - це документ, який описує, які функції потрібно протестувати. Контрольні списки тестування можуть мати різний рівень деталізації.

Тестовий чек-лист складається з двох основних частин [15]:

- список функцій, реалізованих в конкретному продукті, працездатність яких необхідно перевірити;
- список можливих тестів і помилок, які можуть існувати.

Чек-лист тестування програмного забезпечення використовується для розподілу завдань за рівнями кваліфікації, а також для ведення звітності та результатів тестування.

Чек-лист тестування має включати:

- детальний перелік тестів;
- статус перевірки;
- результати перевірки.

Статуси чек-листа:

- Passed – перевірка успішна;

- Failed – знайдені дефекти;
- Blocked – неможливо перевірити;
- In progress – зараз тестується;
- Not run – не перевірений;
- Skipped – пропущений по якійсь причині.

На рисунку 2.1 зображений розроблений чек-лист з відповідними статусами. З рисунка 2.1 видно, що пункт №2 у чек-листі має статус Failed, тобто був знайдений певний дефект. А саме, при перевірці сторінки реєстрації було виявлено, що програма дозволила певному користувачеві зареєструвати акаунт з ім'ям користувача «1», що є неможливим. Тому, коли при реєстрації вводити в поле Ім'я цифру один програма видає повідомлення « Акаунт з даним іменем уже існує», натомість повідомлення, що поле Ім'я може містити лише букви.

№	Name	Status	Comment
1	Register on the site by entering the number 1 instead of the email	pass	Full name - Білик Настя, Phone number: +38(099)9999999, email: 1, password: ********
2	Register on the site by entering the number 1 instead of full name	fail	Doesn't allow to create account with full name 1 because it was already created. Full name - 1, Phone number: +38(099)9999999, email: rattiroinoda@gmail.com, password: ********
3	Register on the site by entering the word instead of phone number	pass	Full name - Білик Настя, Phone number: number, email: rattiroinoda@gmail.com, password: ********.
4	Log in with a Google Account	pass	
5	Go to the Instagram page	pass	
6	Go to the Youtube channel	pass	
7	Check the title by hovering over the tab	pass	
8	Check the search field	pass	
9	Check whether the top menu bar is fixed	pass	
10	Check whether the vertical scroll works correctly.	pass	
11	Check the work of the back to top button	pass	
12	Check the change of banners	pass	
13	Click on the banner at the top of the page	pass	
14	Watch the video	pass	
15	Exit the personal account	pass	
16	Check the tooltips	pass	
17	Check whether contacts or messenger pages open in a new tab	pass	
18	Go to the book page	pass	
19	Check the carousel of books	pass	
20	Click the button "Всі категорії книг"	pass	
21	Go to "Електронні книжки" category using footer	pass	
22	Expand the full description of the bookstore	pass	
23	Collapse the full bookstore description	pass	
24	Add book to cart	pass	
25	Add a book to favorites	pass	

Рисунок 2.1 - Розроблений чек-лист з відповідними статусами

2.4 Тест-кейси

Тестовий кейс – це визначений формат тестування програмного забезпечення, необхідний для перевірки того, чи працює певний додаток/програмне забезпечення чи ні. Тестовий кейс складається з певного набору умов, які необхідно перевірити для тестування програми або програмного забезпечення, тобто при перевірці умов перевіряється, чи відповідає отриманий

результат очікуваному результату чи ні. Тестовий кейс складається з різних параметрів, таких як ідентифікатор, умова, кроки, вхідні дані, очікуваний результат, результат, статус і примітки [11].

Параметри тестового кейсу:

- Назва модуля: тема або назва, яка визначає функціональність тесту;
- Ідентифікатор тестового кейсу: унікальний ідентифікатор, присвоєний кожній умові в тестовому кейсі;
- Ім'я тестувальника: ім'я людини, яка буде проводити тест;
- Сценарій тесту: сценарій тесту надає короткий опис для тестувальника, як невеликий огляд, щоб знати, що потрібно виконати, а також невеликі особливості та компоненти тесту;
- Опис тестового випадку: умова, яку потрібно перевірити для даного програмного забезпечення, наприклад, перевірити, чи працює перевірка тільки чисел для поля введення віку;
- Кроки тесту: кроки, які потрібно виконати для перевірки умови;
- Передумова: умови, які необхідно виконати перед початком процесу тестування;
- Пріоритет тесту: як випливає з назви, надає пріоритет тестовим кейсам, які потрібно виконати в першу чергу або є більш важливими і можуть бути виконані пізніше;
- Тестові дані: вхідні дані, які будуть отримані під час перевірки умов;
- Очікуваний результат тесту: вихідні дані, які потрібно очікувати в кінці тесту;
- Параметри тесту: параметри, призначені для конкретного тестового випадку;
- Фактичний результат: результат, який відображається в кінці тесту;
- Інформація про середовище: середовище, в якому виконується тест, наприклад, операційна система, інформація про безпеку, назва програмного забезпечення, версія програмного забезпечення тощо;
- Статус: статус тесту, наприклад, пройдено, не пройдено, NA тощо;

- Коментарі: зауваження до тесту.

Тестові кейси пишуться в різних ситуаціях:

Перед розробкою: тестові кейси можуть бути написані до фактичного кодування, оскільки це допоможе визначити вимоги до продукту/програмного забезпечення і провести тестування пізніше, коли продукт/програмне забезпечення буде розроблено.

Після розробки: тестові кейси також пишуться безпосередньо після створення продукту/програмного забезпечення або після розробки функції, але до запуску продукту/програмного забезпечення, оскільки це необхідно для тестування роботи конкретної функції.

Під час розробки: тестові кейси іноді пишуться під час розробки, паралельно. Таким чином, щоразу, коли частина модуля/програмного забезпечення розробляється, вона також тестується.

Тестові кейси є одним з найважливіших аспектів програмної інженерії, оскільки вони визначають, як буде проводитися тестування. Тестові кейси проводяться з дуже простої причини, щоб перевірити, чи працює програмне забезпечення чи ні.

Написання тестових кейсів має багато переваг:

- Перевірити, чи відповідає програмне забезпечення очікуванням замовника: тестові кейси допомагають перевірити, чи відповідає певний модуль/програмне забезпечення заданим вимогам чи ні;

- Перевірити відповідність програмного забезпечення умовам: тестові кейси визначають, чи працює певний модуль/програмне забезпечення із заданим набором умов;

- Звузити коло оновлень програмного забезпечення: тестові кейси допомагають звузити потреби в програмному забезпеченні та необхідних оновленнях;

- Краще покриття тестів: тестові кейси допомагають переконатися, що всі можливі сценарії охоплені та задокументовані;

- Для узгодженості у виконанні тестів: тестові кейси допомагають підтримувати узгодженість у виконанні тестів. Добре задокументований тестовий кейс допомагає тестувальнику просто поглянути на нього і почати тестування програми;

- Корисно під час обслуговування: тестові кейси детально описані, що робить їх корисними на етапі супроводу.

Існують певні правила, яких можна дотримуватися при написанні тестових кейсів, і які вважатимуться корисними:

- Простота і ясність: тестові кейси мають бути дуже стислими, чіткими і прозорими. Вони мають бути легкими і простими для розуміння не тільки для вас, але й для інших;

- Підтримка вимог клієнта/замовника/кінцевого користувача має бути унікальною: під час написання тестових кейсів необхідно переконатися, що вони не пишуться знову і знову, і що кожен кейс відрізняється від інших;

- Нуль припущень: тестові кейси не мають містити припущених даних, а також не придумувати функції/модулі, яких не існує;

- Відстежуваність: тестові кейси мають бути відстежуваними для подальшого використання, тому під час написання важливо пам'ятати про це;

- Різні вхідні дані: під час написання тестових кейсів необхідно враховувати всі типи даних;

- Сильна назва: назва модуля має бути зрозумілою під час написання тестового кейсу;

- Мінімальний опис: опис тестового кейсу має бути невеликим, один або два рядки зазвичай вважаються гарною практикою, але він має давати основний огляд належним чином;

- Максимум умов: під час написання тесту потрібно враховувати всі види умов, що підвищує його ефективність;

- Виконання вимог: під час написання тестового кейсу потрібно задовольнити вимоги клієнта/замовника/кінцевого користувача;

- Повторювані результати: тестовий кейс має бути написаний таким чином, щоб він давав один і той же результат;

- Різні техніки: іноді тестування всіх умов може бути неможливим, але використання різних методів тестування з різними тестовими кейсами може допомогти перевірити кожен аспект програмного забезпечення;

- Створюйте тестові кейси з точки зору кінцевого користувача: створюйте тестові кейси, пам'ятаючи про кінцевого користувача, і тестові кейси мають відповідати вимогам замовника;

- Використовуйте унікальний ідентифікатор тестового кейсу: вважається гарною практикою використовувати унікальний ідентифікатор тестового кейсу для тестових кейсів, дотримуючись конвенції про імена для кращого розуміння;

- Додайте належні передумови та післядії: передумови та післяумови для тестових кейсів мають бути вказані правильно і чітко;

- Тестові кейси мають бути багаторазовими: бувають випадки, коли розробник оновлює код, тоді тестувальникам потрібно оновити тестові кейси, щоб відповідати вимогам, які змінюються;

- Вкажіть точний очікуваний результат: вказуйте точний очікуваний результат, який говорить про те, що буде результатом конкретного кроку тестування.

Існують різні типи тестових кейсів:

- Тестовий кейс функціональності: тестовий кейс функціональності полягає в тому, щоб визначити, чи працює інтерфейс програмного забезпечення безперебійно з рештою системи та її користувачами чи ні. При перевірці цього тестового випадку використовується тестування "чорного ящика", оскільки перевіряємо все ззовні, а не зсередини;

- Одиничний тестовий кейс: в одиничному тестовому кейсі тестується окрема частина або окрема одиниця програмного забезпечення. Тут тестується кожен модуль/окрема частина, і ми створюємо окремий тестовий кейс для кожного модуля;

- Тестовий кейс інтерфейсу користувача: тест інтерфейсу користувача - це коли тестується кожен компонент інтерфейсу користувача, з яким користувач буде контактувати. Це для того, щоб перевірити, чи виконані вимоги користувача до компонентів інтерфейсу чи ні;

- Тестовий приклад інтеграції: інтеграційне тестування - це коли всі блоки програмного забезпечення об'єднуються, а потім вони тестуються. Це перевірка того, що кожен компонент та його підрозділи працюють разом без будь-яких проблем;

- Тестовий кейс продуктивності: тестовий кейс продуктивності допомагає визначити час відгуку, а також загальну ефективність системи / програмного забезпечення. Це для того, щоб побачити, чи буде додаток відповідати реальним очікуванням;

- Тестовий кейс бази даних: також відоме як внутрішнє тестування або тестування даних. Виконується перевірка, чи все працює правильно з базою даних. Виконуються тестові кейси для таблиць, схеми, тригерів тощо;

- Тест безпеки: допомагає визначити, що додаток обмежує дії, а також дозволи, де це необхідно. Шифрування та аутентифікація вважаються основними цілями тестового кейсу безпеки. Тест безпеки проводиться для захисту та збереження даних програмного забезпечення;

- Тестовий кейс юзабіліті: також відомий як тестовий кейс користувацького досвіду, він перевіряє, наскільки зручним або простим у використанні буде програмне забезпечення. Юзабіліті-тести розробляються командою користувачів і виконуються командою тестувальників;

- Тестовий кейс прийняття користувачем: тестовий кейс прийняття користувачем готується командою тестування, але користувач/клієнт проводить тестування та огляд, якщо він працює в реальному середовищі.

На рисунках 2.2 − 2.7 зображено частину розроблених тест-кейсів, які базуються на різних техніках тест-дизайну. Всього було розроблено 15 тест-кейсів, їх в повному обсязі можна буде переглянути за посиланням в джерелах [12].

ID 1	
Summary	Add the book to the cart
Precondition	The user is authorized on the site nashformat.ua with the following test data: amina.melnychuk@gmail.com password: ********.
Test design technique	State transition
Steps	1. Go to the section "Художня література". 2. Go to the subcategory "Іноземна класика". 3. Click on the first book. 4. Click the "Add to cart" button. 5. Click the button "Продовжити покупки"in the pop-up window
Expected result	The book is added to cart and number 1 should appear near the cart image in the upper right corner.
Attachment:	

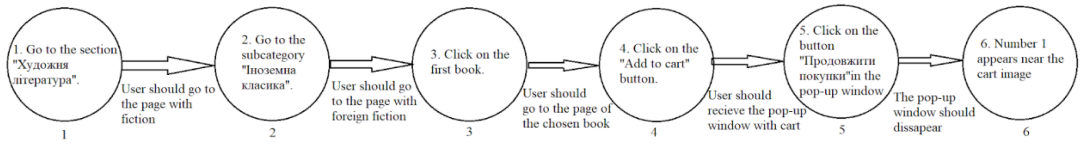


Рисунок 2.2 – Тест-кейс про додавання книги до кошика, створений на основі техніки тест дизайну State transition

ID 2					
Summary	Fill in the field Name and Surname in the personal information				
Precondition	1) The user is authorized on the site nashformat.ua with the following test data: amina.melnychuk@gmail.com password: ********. 2) The total number of characters in the Name and Last Name field can be from 6 to 12				
Test design technique	Boundary value				
	invalid group		valid group		invalid group
	-	5	6	12	13 `+
Steps					
	case 1	Enter data in the field name and surname, which have a total of 5 characters			
	Expected result	Must issue an error			
	case 2	Enter data in the field name and surname, which have a total of 6 characters			
	Expected result	Data must be saved			
	case 3	Enter data in the field name and surname, which have a total of 12 characters			
	Expected result	Data must be saved			
	case 4	Enter data in the field name and surname, which have a total of 13 characters			
	Expected result	Must issue an error			

Рисунок 2.3 – Тест-кейс про заповнення полів Ім’я та Прізвище в Персональній інформації користувача, створений на основі техніки тест-дизайну Boundary value

ID 4							
Summary	Enter the e-mail for user registration						
Precondition	Go to the site nashformat.ua						
Test design technique	Decision table						
Steps							
	Case	**1**	**2**	**3**	**4**	**5**	**6**
	Should contain @	0	1	0	0	1	1
	Should contain .com	0	1	0	1	1	0
	Written in lower case	0	1	1	0	0	1
	Written in English	0	1	1	1	0	0
	Result	Enter e-mail	Success	Write the correct e-mail, should contain @ and .com	Write the correct e-mail, should be written in lower case and contain @	Write the correct e-mail, should be written in English and in lower case	Write the correct e-mail, should contain .com and be written in English

Рисунок 2.4 – Тест-кейс про введення електронної адреси для реєстації акаунта користувача, створений на основі техніки тест дизайну Decision table

ID 5		
Summary	Change the price in the price filter on the Book page	
Precondition	The user is authorized on the site nashformat.ua with the following test data: amina.melnychuk@gmail.com password: ********.	
Test design technique	Equivalence partioning	
	Affordable price from 185 to 468	
	valid group	invalid group
	185	184
	468	469
	359	-185
	400	50
	285	100
Steps	case 1	Enter the minimum price 359
	Expected result	Books are filtered with a price from 359
	case 2	Enter the minimum price - 185
	Expected result	The price filter should not save data and should keep the original values

Рисунок 2.5 – Тест-кейс про зміну ціни в фільтрах на сторінці Книги, створений на основі техніки тест-дизайну Equivalence partioning

ID 6	
Summary	Registration on the website nashformat.ua
Precondition	Go to nashformat.ua
Test design technique	Exploritary testing
Steps	1) Click the "Login" button in the upper right corner.
	2) Change the "Login" tab to the "Registration" tab in the pop-up window
	3) Enter last name and first name: Melnychuk Amina
	4) Enter phone number: +38 (093) 023-11-14
	5) Enter e-mail address: amina.melnychuk@gmail.com
	6) Enter password: ********
	7) Click the "Register" button.
Expected result	The user is registered on the site

Рисунок 2.6 – Тест-кейс про реєстрацію на сайті, створений на основі техніки тест-дизайну Exploritary testing

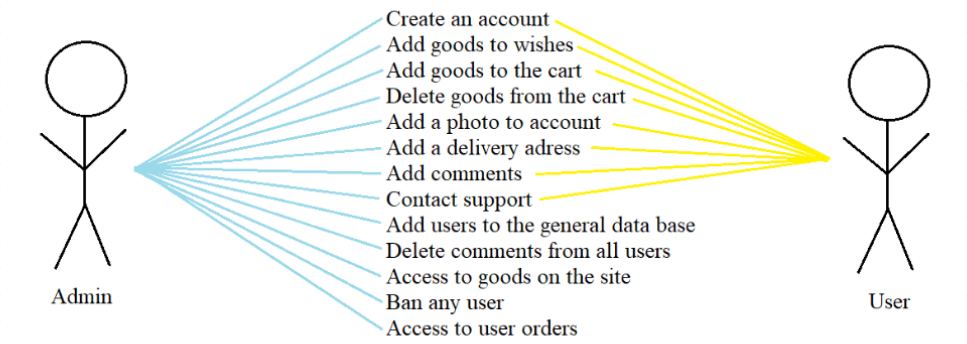
ID 11	
Summary	Explore the differences between user and admin possibilities
Precondition	Go to site nashformat.ua.
Test design technique	Use-case diagram
Attachment:	
Expected result	User shouldn't have the access to the functions of admin

Рисунок 2.7 – Тест-кейс про дослідження різниці можливостей користувача та адміністратора, створений на основі техніки тест-дизайну Use-case diagram

2.5 Баг-репорти

Під час процесу контролю якості, коли виявлено помилку, її потрібно задокументувати та надіслати розробникам для виправлення. Враховуючи, що програмне забезпечення в сучасному цифровому середовищі є надзвичайно складним, багаторівневим і багатофункціональним, більшість конвеєрів контролю якості генерують безліч помилок.

Крім того, розробники часто працюють над кількома проектами одночасно, а це означає, що вони мають значну кількість помилок, які потребують уваги. Їм доводиться працювати під значним тиском, і без належних ресурсів вони можуть бути перевантажені.

Природно, що QA витрачають багато часу на дослідження того, як повідомити про помилку таким чином, щоб це принесло користь розробникам і допомогло їм швидко та ефективно налагоджувати.

Добре структуровані та адекватно деталізовані звіти про помилки є одним з таких ресурсів. Хороші звіти про помилки точно вказують розробникам, що саме потрібно виправити, і допомагають їм зробити це швидше. Це запобігає затримці випуску програмного забезпечення, пропонуючи швидший час виходу на ринок без шкоди для якості.

Переваги хорошого баг-звіту:

- Хороший звіт про ваду містить всю важливу інформацію про ваду, яка може бути використана в процесі налагодження;
- Допомагає провести детальний аналіз помилки;
- Дає краще уявлення про помилку і допомагає знайти правильний напрямок і підхід до налагодження;
- Заощаджує кошти та час, допомагаючи налагодити на більш ранній стадії;
- Запобігає потраплянню помилок у виробництво і погіршенню якості роботи кінцевих користувачів;
- Діє як керівництво, що допомагає уникнути тих самих помилок у майбутніх випусках;

- Інформує всі зацікавлені сторони про помилку, допомагаючи їм вжити коригувальних заходів.

Елементи ефективного баг-звіту [16, 17]:

Вивчаючи, як створити баг-звіт, почніть з питання: Що баг-звіт повинен сказати розробнику?

Баг-звіт повинен відповідати на наступні питання:

- У чому полягає проблема?
- Як розробник може відтворити проблему (побачити її на власні очі)?
- Де в програмному забезпеченні (яка веб-сторінка або функція) з'явилася проблема?
- Яке середовище (браузер, пристрій, ОС), в якому виникла проблема?

Ефективне повідомлення про ваду повинно містити наступну інформацію:

1. Назва/Ідентифікатор помилки;
2. Середовище;
3. Кроки для відтворення помилки;
4. Очікуваний результат;
5. Фактичний результат;
6. Візуальний доказ (скріншоти, відео, текст) помилки;
7. Серйозність/Пріоритет.

Назва повинна містити короткий опис вади. Наприклад, "Спотворений текст у розділі поширених запитань на домашній сторінці <назва>".

Присвоєння помилці ідентифікатора також допомагає полегшити її ідентифікацію.

Дефект може з'являтися у певному середовищі, але не в інших. Наприклад, баг з'являється під час запуску веб-сайту в Firefox, або додаток не працює лише під час запуску на iPhone X. Такі баги можна виявити лише за допомогою крос-браузерного тестування або тестування на різних пристроях.

Повідомляючи про помилку, QA повинні вказати, чи спостерігається вона в одному або декількох конкретних середовищах. Використовуйте шаблон нижче для конкретизації:

- Тип пристрою: Апаратне забезпечення та конкретна модель пристрою;
- Операційна система: Назва та версія ОС;
- Тестувальник: Ім'я тестувальника, який виявив помилку;
- Версія програмного забезпечення: Версія програмного забезпечення, яке тестується, і в якому з'явилася помилка;
- Сила з'єднання: Якщо помилка залежить від інтернет-з'єднання (4G, 3G, WiFi, Ethernet), вкажіть його силу на під час тестування;
- Швидкість відтворення: Кількість разів, коли баг був відтворений, із зазначенням точних кроків, задіяних у кожному відтворенні.

Чітко пронумеруйте кроки від першого до останнього, щоб розробники могли швидко і точно виконати їх, щоб побачити помилку на власні очі.

Очікуваний результат баг-звіту описує, як програмне забезпечення повинно функціонувати в заданому сценарії. Розробник дізнається, які вимоги висуваються до очікуваного результату. Це допомагає йому оцінити ступінь, до якого баг порушує користувацький досвід.

Опишіть ідеальний сценарій кінцевого користувача і спробуйте запропонувати якомога більше деталей.

Детально опишіть, що насправді робить баг і як він спотворює очікуваний результат.

Конкретність у цьому розділі буде дуже корисною для розробників. Чітко підкресліть, що саме йде не так. Надайте додаткові деталі, щоб вони могли почати досліджувати проблему з урахуванням усіх змінних

Скріншоти, відеозаписи лог-файлів повинні бути прикріплені, щоб наочно зобразити появу помилки. Залежно від характеру помилки, розробнику можуть знадобитися відео, текст і зображення.

Кожній помилці має бути присвоєно рівень серйозності та відповідний пріоритет. Це показує, наскільки сильно помилка впливає на систему, і, в свою чергу, як швидко її потрібно виправити.

Рівні серйозності помилки (Severity):

- Низький: помилка не призведе до помітного збою в роботі системи;

- Незначний: Призводить до неочікуваної або небажаної поведінки, але не настільки, щоб порушити роботу системи;
- Серйозний: Помилка здатна вивести з ладу значну частину системи;
- Критичний: помилка здатна спричинити повне вимкнення системи.

Рівні пріоритетності помилок (Priority):

- Низький: помилка може бути виправлена пізніше. Пріоритет мають інші, більш серйозні помилки;
- Середній: Помилка може бути виправлена у звичайному процесі розробки та тестування;
- Високий: помилка повинна бути вирішена якомога швидше, оскільки вона негативно впливає на систему і робить її непридатною для використання до тих пір, поки її не буде вирішено.

На рисунку 2.8 зображений життєвий цикл дефекту:

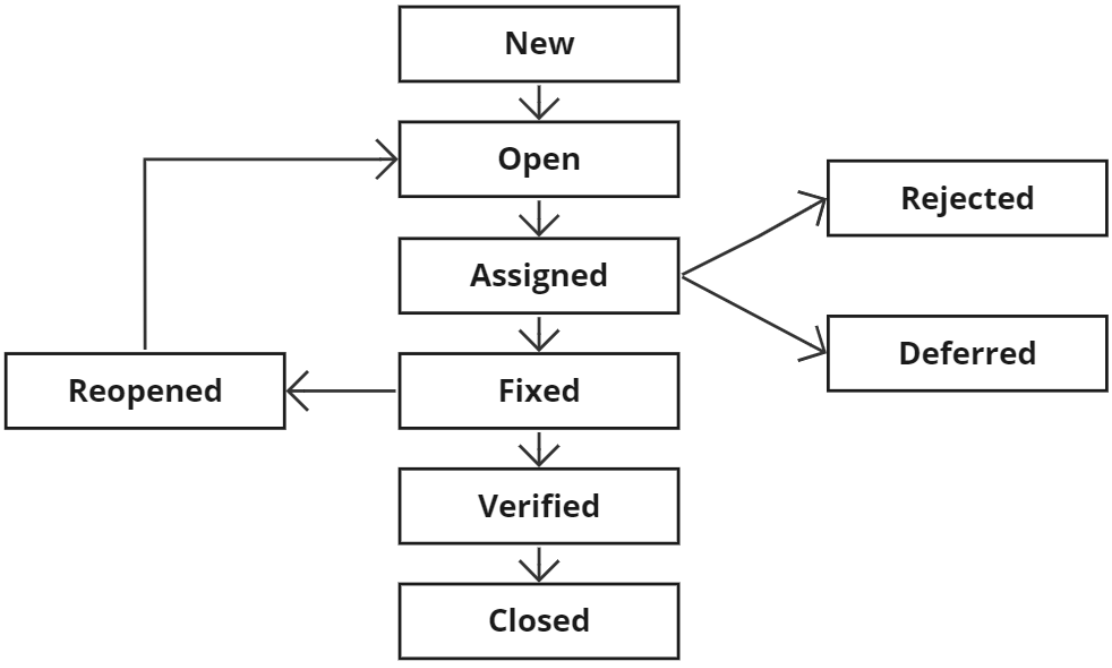


Рисунок 2.8 –Життєвий цикл дефекту

Спершу кожен баг-репорт має статус New, тобто щойно створений. Далі він отримує статус Open(відкритий), коли хтось переглядає даний файл та закріплює

даний файл за кимось змінюючи статус на Assigned. Далі життя дефекту може мати три шляхи: Rejected, тобто відхилений; Deferred, тобто перенесений на інший спринт; Fixed, тобто виправлений. Після повторної перевірки дефект може мати статус Reopened, якщо під час повторного тестування виявилось, що помилка залишилась, або Verified, тобто помилка була успішно виправлена. І завершальним є статус Closed, який означає, що баг-репорт був закритим.

Розрізняють також наступні види помилок: Bug, Defect, Error, Failure.

Bug – це помилка в продукті, яка викликає його непередбачуване функціонування. Виявлена під час тестування.

Defect – це помилка, виявлена після релізу (вихід продукту в live).

Error – це помилка, яка виникає внаслідок некоректних дій користувача (приклад: помилка 404 на стороні клієнта)

Failure (падіння) – це відмова роботи програми (приклад: помилки 5xx на стороні сервера).

Нижче на рисунках 2.9 – 2.13 зображені баг-репорти за знайденими помилками.

ID - 1	
Summary	The user can register on the site with full name 1
Precondition	Go to site nashformat.ua
Steps	1) 1) You click the button "Вхід"
	2) In the pop-up window go to "Реєстрацію"
	3) Enter full name: 1
	4) Enter phone number, email and password: +38(099)9999999, rattiroinoda@gmail.com, ********
	5) Click the button "Зареєструватися"
Expected result	The user will not be able to register an account with name 1 and must receive a message about the invalidity of the entered data in the name field
Actual result	The user receives a message that the user with this name already exists, so it is possible to register with a number instead of a full name
Severity	Major
Priority	Medium
Type	Logical
Assigned to	Dev Eva Silko
Environment:	Windows 11, Google Chrome Version 100.0.4896.88
Attachment:	https://drive.google.com/file/d/1dKQmornIP8g7c568CFVSB46aZmSP5ckN/view?usp=sharing

Рисунок 2.9 – Баг-репорт про можливість реєстрації користувача з Іменем 1

ID - 2	
Summary	The user can add a delivery address without specifying a new post office
Precondition	The user is logged in on the site nashformat.ua: tanagulka@gmail.com password: ********.
Steps	1) Hower the mouse over the inscription "Вітаємо, Настя".
	2) In the pop-up window choose "Адреси доставки".
	3) Click the button "Додати адресу".
	4) Click on the image "Нова пошта".
	5) Uder the images choose "Відділення-відділення".
	6) Choose in field "Місто" needed city from the drop down list: Вінниця.
	7) Enter nothing in the field "Відділення"
	8) Click the button "Зберегти".
Expected result	The user should be notified of the lack of data in the field "Відділення"
Actual result	The program automatically selects any branch and adds a delivery address
Severity	Major
Priority	Medium
Type	Functional
Assigned to	Dev Eva Silko
Environment:	Windows 11, Google Chrome Version 100.0.4896.88
Attachment:	https://drive.google.com/file/d/1ThDWhmGByusWtxKmTyeaJRLuCozeHub9/view?usp=sharing

Рисунок 2.10 – Баг-репорт про можливість залишити поле відділення пустим при додаванні адреси доставки

ID - 3	
Summary	The name of the subcategory of books overlaps the number of books
Precondition	The user is authorized on the site nashformat.ua with the following test data: amina.melnychuk@gmail.com password: ********.
Steps	1) Click on "Книги" in the top menu
	2) Choose "Історія. Військова справа"
Expected result	The user should see clearly written all the names of the subcategories and the number of books on the left side of the page
Actual result	The inscription Військова справа. Спецслужби overlaps the number og the books in this subcategory
Severity	Minor
Priority	Low
Type	UI
Assigned to	Dev Eva Silko
Environment:	Windows 11, Google Chrome Version 100.0.4896.88
Attachment:	https://drive.google.com/file/d/1ilxTslF-5zwu1gu9D_RNjY0yI0MEANMI/view?usp=sharing

Рисунок 2.11 – Баг-репорт про те, що назви підкатегорій книг перекривають кількість книг

ID - 4	
Summary	Wishes counters on books don't work correctly
Precondition	The user is authorized on the site nashformat.ua with the following test data: amina.melnychuk@gmail.com password: ********.
Steps	1) Scroll down the home page
	2) Click on the first bool in section "Нові книги"
	3) Add book to wishes many times by clicking the heart
Expected result	The counter should add and subtract 1 to the number of wishes
Actual result	The counter only subtrects 1 every time you click the heart
Severity	Major
Priority	Medium
Type	Functional
Assigned to	Dev Eva Silko
Environment:	Windows 11, Google Chrome Version 100.0.4896.88
Attachment:	https://drive.google.com/file/d/11xUpUnbLP1dhXkBRXvsMiFjRgD0qBEoL/view?usp=sharing

Рисунок 2.12 – Баг-репорт про те, що кнопки додати до побажань працюють некоректно

ID - 5	
Summary	Drop down list works incorect
Precondition	The user is authorized on the site nashformat.ua with the following test data: amina.melnychuk@gmail.com password: ********.
Steps	1) Go to section "Книги"
	2) Choose category "Електронні книги"
	3) On the left side in the filter menu close and open drop down lists
Expected result	Drop down list should open smoothly
Actual result	Drop down list opens sharply
Severity	Major
Priority	Medium
Type	Functional
Assigned to	Dev Eva Silko
Environment:	Windows 11, Google Chrome Version 100.0.4896.88
Attachment:	https://drive.google.com/file/d/1UiSBNmFvi785CaYcv7-5AFSJT1dMaJB6/view?usp=sharing

Рисунок 2.13 – Баг-репорт про те, що випадаючий список працює некоректно

2.6 Автоматизовані тести

Автоматизоване тестування – це частина процесу тестування, що використовує програмні засоби для виконання тестів та перевірки результатів.

Selenium IDE – це інструмент в Google Chrome, Mozilla Firefox, який дозволяє записувати та відтворювати тести в браузерах.

На рисунку 2.14 представлене зображення створених автоматизованих тестів в Selenium IDE на основі розроблених вище тест-кейсів: додати книгу до бажань, додати адресу доставки, додати книгу в кошик, змінити ціну в фільтрах на сторінці Книги, ввести електрону пошту при реєстрації аканту користувача, збільшити кількість книг в кошику, залогінитись на сайті, зареєструватись на сайті.

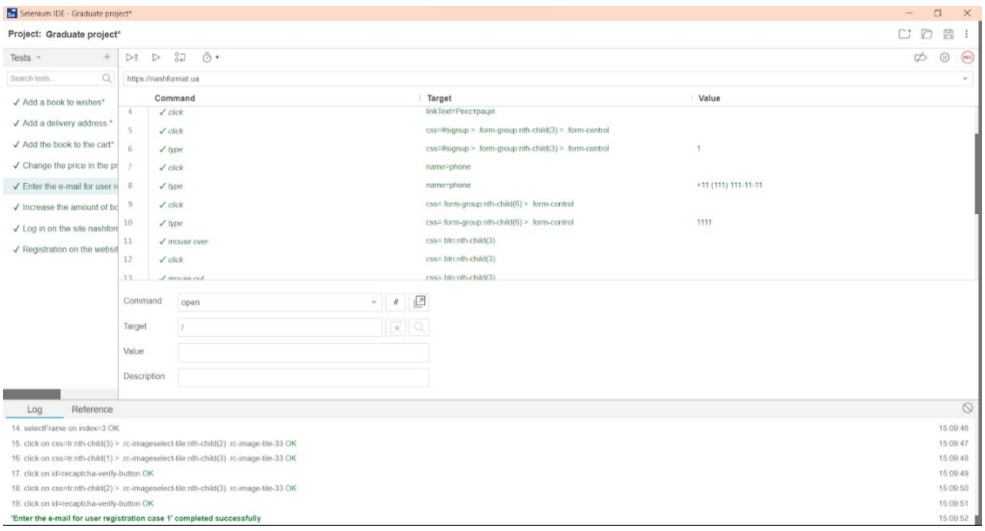


Рисунок 2.14 − Створені автоматизовані тести в Selenium IDE

2.7 Розробка алгоритмів для покриття автоматизованими тестами WEB-застосунку

Алгоритм – це покроковий опис певного процесу, який має визначений початок та кінець. Якщо алгоритм не має завершення, це означає що у процесі розробки виникла помилка. Алгоритм спрощує роботу системи або WEB-застосунка, полегшуючи етап розробки, особливо логіку в певному модулі.

Алгоритм можна описати такими способами:

– лінгвістичний;

– схематичний.

Лінгвістичний спосіб опису передбачає опис алгоритму словами. Основним недоліком цього підходу є те, що інформація може забуватися, тому програмісту доводиться постійно уточнювати деталі.

Схематичний спосіб опису передбачає використання структурних блоків для представлення алгоритму. Основною перевагою цього методу є те, що достатньо один раз розробити схему і надати її програмісту.

Отже, для розробки автоматизованих тестів можна розробити декілька сценаріїв дій користувача на сайті лінгвістичним способом.

На рисунку 2.15 подано схему алгоритму перевірки введення початкових даних.

І сценарій

Алгоритм дій користувача у випадку реєстрації на сайті складається з таких кроків:

Крок 1. Користувач заходить на сторінку сайту.

Крок 2. Користувач натискає на кнопку «Вхід» у верхньому правому кутку.

Крок 3. Користувач переходить на вкладку Реєстрації натиснувши кнопку «Реєстрація».

Крок 4. Користувач вводить своє ПІБ у поле «Прізвище Ім’я».

Крок 5. Користувач вводить свій номер телефону в поле «+38 (XXX) XXX-XX-XX».

Крок 6. Користувач вводить свою електронну адресу в поле «E-mail».

Крок 7. Користувач створює свій унікальний пароль у поле «Пароль».

Крок 8. Користувач натискає кнопку «Зареєструватись».

Крок 9. Користувач є успішно зареєстрованим, якщо всі дані введено правильно.

За таким сценарієм буде перевірятись чи приймає система невалідні дані для реєстрації.

ІІ сценарій

Алгоритм у випадку, коли користувач хоче придбати книгу:

Передумова: Користувач є зареєстрованим на сайті.

Крок 1. Користувач натискає кнопку «Вхід».

Крок 2. Користувач вводить електронну пошту, на яку він раніше зареєстрував акаунт у поле «E-mail».

Крок 3. Користувач вводить свій пароль в поле «Пароль».

Крок 4. Користувач натискає кнопку «Увійти».

Крок 5. Користувач наводить на випадаючий список «Книги».

Крок 6. Користувач обирає у випадаючому списку потрібний відділ та натискає на нього, наприклад «Художня література».

Крок 7. Користувач пролистує сторінку нижче.

Крок 8. Користувач обирає книгу та натискає на неї, наприклад Василь Симоненко «Задивляюсь у твої зіниці».

Крок 9. Користувач натискає кнопку «Купити».

Крок 10. Користувач натискає кнопку «Оформити» у спливаючому вікні.

Крок 11. Користувач автоматично потрапляє на сторінку оформлення замовлення.

Крок 12. Користувач обирає, що він є фізичною особою.

Крок 13. Користувач вводить свій номер телефону в поле «Мобільний телефон».

Крок 14. Користувач вводить свою електронну пошту в поле «E-mail».

Крок 15. Користувач вводить своє ПІБ у поле «Прізвище Ім’я».

Крок 16. Користувач натискає кнопку «Далі».

Крок 17. Користувач на сторінці «Доставка» обирає в випадаючому меню «Країна» потрібну країну, наприклад «Україна».

Крок 18. Користувач обирає у випадаючому меню «Оберіть спосіб доставки» потрібну службу доставки, наприклад «Нова пошта».

Крок 19. Користувач обирає варіант «Відділення-Відділення».

Крок 20. Користувач обирає місто доставки з випадаючого меню «Місто».

Крок 21. Користувач обирає відділення з випадаючого меню «Відділення».

Крок 22. Користувач додає коментар за потреби в поле «Коментар до замовлення».

Крок 23. Користувач натискає кнопку «Далі».

За таким сценарієм потім користувач обирає спосіб оплати, оплачує та отримує створене замовлення. В розглянутому випадку цього робити не будемо, адже тестування проходить на існуючому довільному сайту і ми не хочемо приностити незручності сервісу створюючи недійсні замовлення. Таким сценарієм буде перевірятись реакція системи введення недійсної чи неправильної інформації у поля.

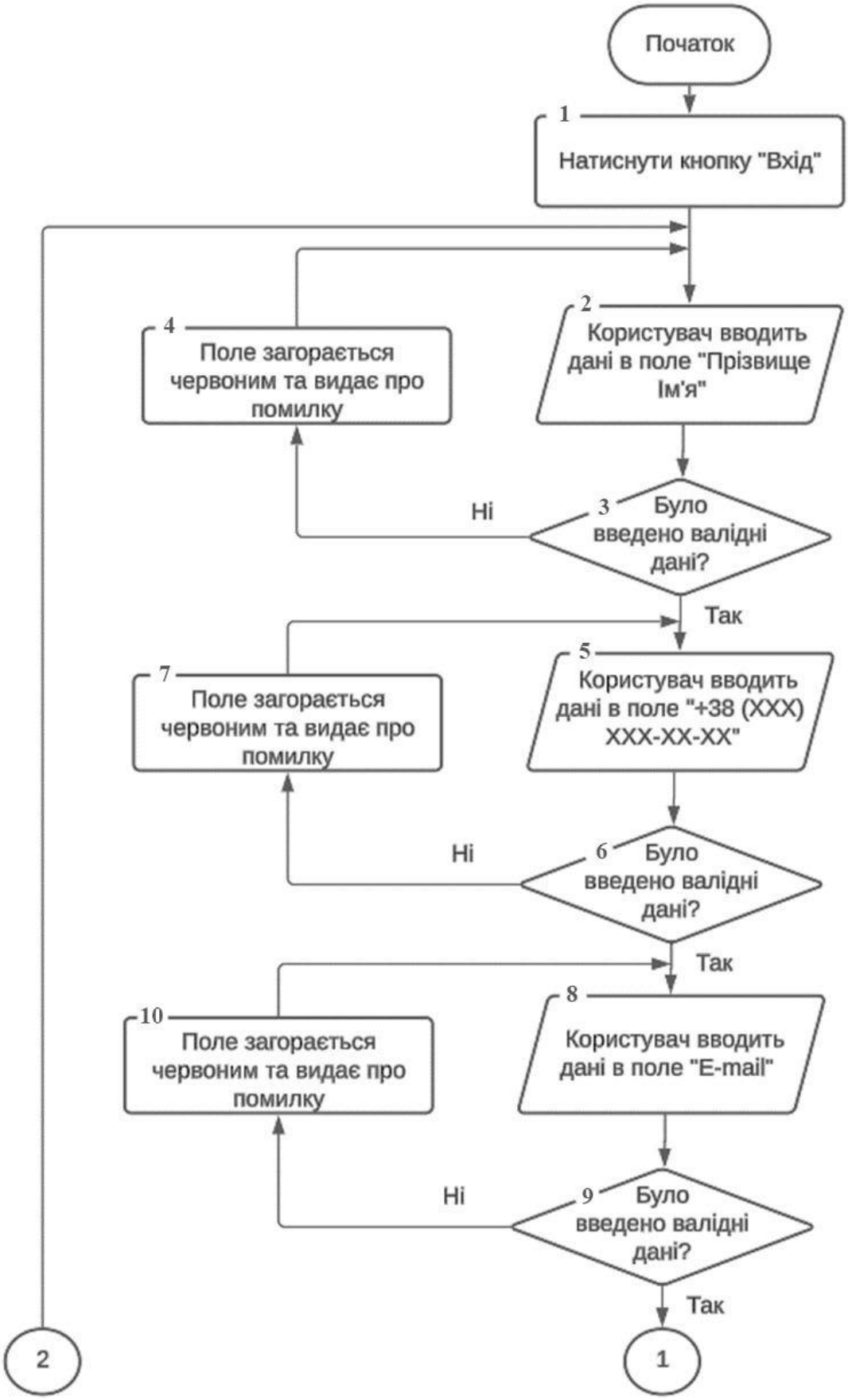


Рисунок 2.15 – Схема алгоритму перевірки введення початкових даних

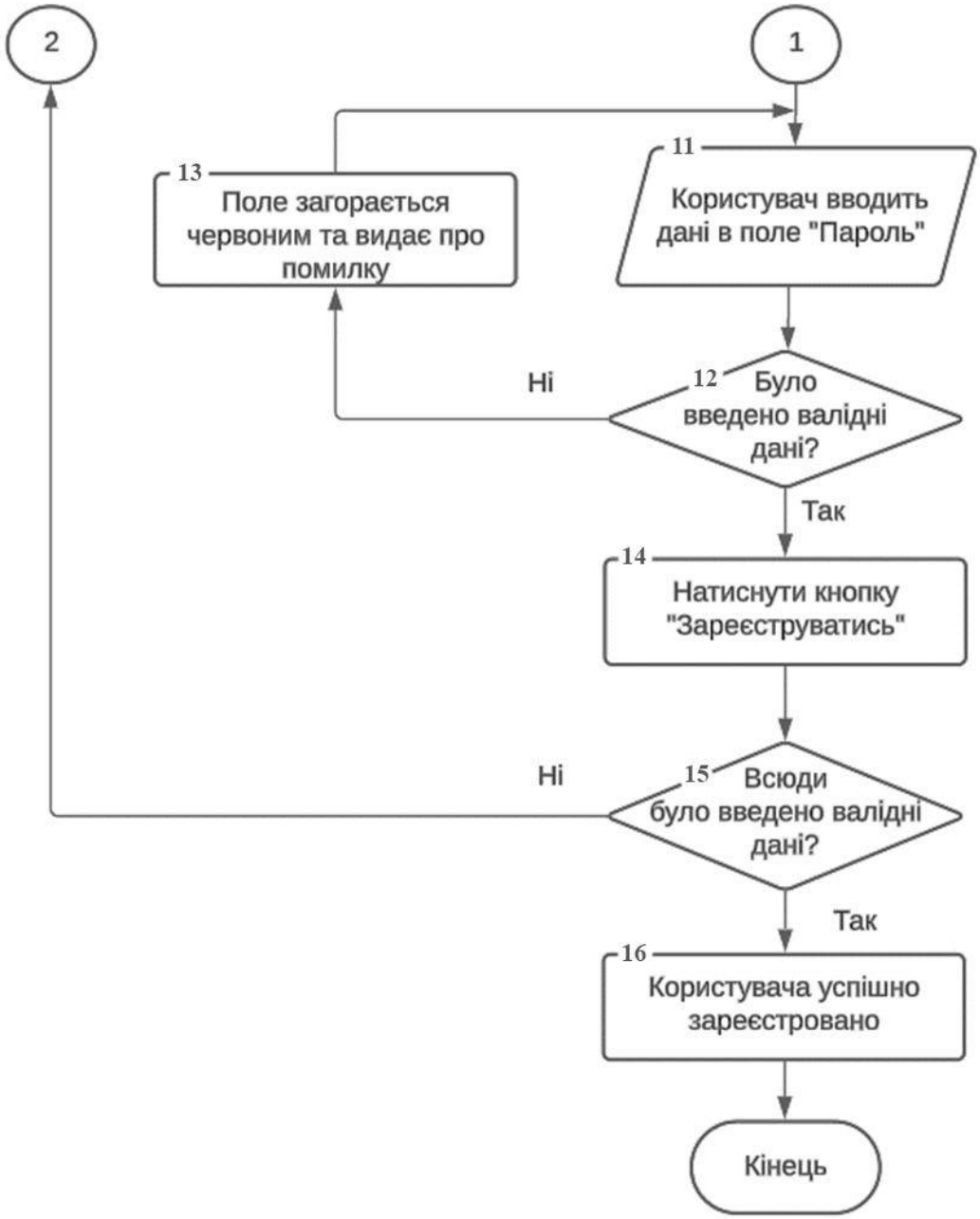


Рисунок 2.15, аркуш 2

2.8 Висновок

В цьому розділі були проаналізовані та сформульовані основні артефакти тестового модуля забезпечення аналізу якості WEB-застосунку.

Розроблено 10 сценаріїв для фунціоналу та для інтерфейсу WEB-застосунку, які дають стисле пояснення певного функціоналу або його можливостей з точки

зору кінцевого користувача. Розроблені сценарії було використано для створення вимог до програмного забезпечення.

Створений чек-лист окреслює функціонал, який підлягає перевірці на певному етапі та допомагає визначити рівень якості функціоналу на момент перевірки. Також було створено ряд тест-кейсів із використанням різних тестових підходів та технік тест-дизайну. Тестові кейси є одним з найважливіших аспектів програмної інженерії, оскільки вони визначають, як буде проводитися тестування.

За результатами тестування на основі чек-листів та тест-кейсів, було визначено ряд помилок, на основі яких були створені відповідні баг-репорти. Кожен баг-репорт має чітку структуру та дає мождивість зрозуміти в якому місці функціоналу проблема, її суть, описує кроки відтворення, а також критичність та важливість вирішення.

Також було розглянуто приклад автоматизованих тестів за допомогою інструменту Selenium IDE. Такий інструмент записує дії користувача на сайті та надалі відтворює їх. Таким чином ці тести допомогають перенести тестування частини функціоналу в автоматичний режим. Такий вид автотестів є легким в реалізації, адже не вимогає знянь мов програмування. До переваг можна віднести також те, що вони полегшують навантаження на тестувальника та дають змогу зекономити час. Для більш поглибленого тестування, в деяких випадках перевірки коду доцільніше використовувати автоматичні тести, створені за допомогою коду.

Всі ці кроки допомагають налагодити ефективний процес тестування функціоналу, показати рівень якості розробленого продукту на момент проведення перевірки, визначити проблеми, вирішення яких покращує роботу WEB-застосунку, що прямо впливає на конверсію сайту, а отже на прибуток замовника.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДУЛЯ ТЕСТУВАННЯ WEB-ЗАСТОСУНКІВ

3.1 Існуючі платформи для розробки програмного модуля

На сьогодні існує багато платформ, на яких можна розробити програмний модуль тестування. Щоб вибрати найбільш вдалий варіант, розглянемо популярні платформи, такі як Selenium, Postman, JMeter та TestNG.

Selenium призначений для тестування сайтів і WEB-застосунків на різних операційних системах і браузерах. Його перевага в тому, що тестувальники можуть обирати свою мову програмування для написання тестів. Серед основних: Python, Perl, PHP, Java, C # та ін [18].

Postman − програма призначена для створення і тестування роботи програмних інтерфейсів додатків (API) і WEB-сайтів, а також для відправки запитів на сервер. Завдяки графічному інтерфейсу можна легко налаштовувати всі необхідні дані для проведення тестів [19].

JMeter створювався для тестування WEB-застосунків, але сьогодні його функціонал дозволяє проводити тестування навантаження для таких сполук як FTP, HTTP, JDBC, POP3, LDAP та ін. З його допомогою можна створити групу запитів відразу з декількох ПК. Результати тестів оформляються з використанням інфографіки [20].

TestNG написаний на Java фреймфорк для автоматизації тестів, поєднує в собі функціонал JUnit і NUnit поряд із новими функціями і багатопотоковим тестуванням. Простий у використанні, цей інструмент забезпечує підтримку основних типів тестування, включаючи функціональне, інтеграційне тощо [21].

У таблиці 3.1 представлено основні відмінності між платформами для автоматизації тестування.

Таблиця 3.1 – Порівняння платформ для автоматизації тестування

Назва критерію	**Selenium**	**Postman**
Основний сценарій використання	Автоматизація браузерів та тестування WEB-застосунків	Тестування API
Підтримувані протоколи	HTTP, HTTPS	HTTP, HTTPS
Мова сценаріїв	Java, Python, C#, Ruby, JavaScript	JavaScript
Взаємодія з графічним інтерфейсом	+	-
Управління тестуванням	Обмежене	Обмежене
Інтеграція CI/CD	+ (Jenkins, Maven, etc.)	+ (Jenkins, Newman, etc.)
Асинхронне тестування	-	Обмежене
Паралельне виконання	+	+
Звітність про тестування	Базові звіти	Базові звіти
Ліцензія	Open source (Apache License 2.0)	Free

Продовження таблиці 3.1

Назва критерію	**JMeter**	**TestNG**
Основний сценарій використання	Тестування продуктивності та навантаження	Тестовий фреймворк для Java
Підтримувані протоколи	HTTP, HTTPS, FTP, JDBC, JMS	Залежить від інтегрованих інструментів
Мова сценаріїв	Java, Groovy	Java
Взаємодія з графічним інтерфейсом	-	-
Управління тестуванням	Обмежене	Комплексні можливості управління тестуванням
Інтеграція CI/CD	+ (Jenkins, Maven, etc.)	+ (Jenkins, Maven, etc.)
Асинхронне тестування	+	+
Паралельне виконання	+	+
Звітність про тестування	Детальні звіти	Детальні звіти
Ліцензія	Open source (Apache License 2.0)	Open source (Apache License 2.0)

На основі аналізу даних з таблиці 3.1, можна визначити такі переваги на варіанти використання платформ:

1. Selenium найкращий для автоматизації браузерів та тестування WEB-застосунків.

Переваги:

- Автоматизує WEB-браузери та імітує взаємодію з користувачем;
- Підтримує декілька мов програмування (Java, Python, C# тощо);
- Добре інтегрується з різними тестовими фреймворками (наприклад, TestNG, JUnit);
- Велика спільнота та обширна документація.

Варіанти використання:

- Функціональне тестування WEB-застосунків;
- Регресійне тестування;
- Кросбраузерне тестування.

2. Postman найкращий для тестування API.

Переваги:

- Простий у використанні інтерфейс для проектування, тестування та документування API;
- Підтримує автоматизацію тестування API за допомогою Newman (інструмент командного рядка);
- Можна створювати складні робочі процеси за допомогою тестових скриптів, написаних на JavaScript;
- Детальна перевірка та моніторинг відповідей API.

Приклади використання:

- Тестування RESTful API;
- Тестування інтеграції API;
- Автоматизоване тестування API в CI/CD пайплайнах.

3. JMeter найкращий для тестування продуктивності та навантаження.

Переваги:

- Імітує високі навантаження на сервери, мережі або об'єкти;

- Підтримує різні протоколи (HTTP, HTTPS, FTP, JDBC, JMS);
- Надає вичерпні звіти та графіки продуктивності;
- Розширюється за допомогою плагінів.

Варіанти використання: навантажувальне тестування WEB-серверів і серверів додатків; стрес-тестування; вимірювання продуктивності системи при різних умовах навантаження.

4. TestNG найкраще підходить для тестових фреймворків, часто використовується в поєднанні з Selenium.

Переваги:

- Призначений для конфігурації тестів, запуску декількох тестів і створення звітів;
- Підтримує анотації (@Test, @BeforeMethod, @AfterMethod тощо) для кращого контролю виконання тестів;
- Добре інтегрується з інструментами збірки, такими як Maven, та інструментами CI/CD, такими як Jenkins;
- Паралельне виконання і тестування на основі даних;

Варіанти використання:

- Модульне тестування;
- Інтеграційне тестування;
- Управління тестуванням для тестів на основі Selenium.

На основі аналізу даних з табл. 3.1, стає очевидним, що найкращими варіантом для створення програмного модуля тестування WEB-застосунків є Selenium.

3.2 Обґрунтування вибору мови програмування

Найпопулярнішими мовами програмування на платформі Selenium є Python, Perl, PHP, Java та C#.

Python − це високорівнева інтерпретована мова програмування, відома своєю простотою, читабельністю та універсальністю. Python підтримує декілька

парадигм програмування, включаючи процедурне, об'єктно-орієнтоване та функціональне програмування, що робить її популярним вибором для широкого спектру додатків [22].

Perl − це високорівнева інтерпретована мова програмування, відома своїми можливостями обробки тексту, універсальністю та підтримкою декількох парадигм програмування. Perl часто використовується для написання сценаріїв, системного адміністрування, веб-розробки та мережевого програмування [23].

PHP (Hypertext Preprocessor − препроцесор гіпертексту) − це широко використовувана мова сценаріїв з відкритим вихідним кодом, розроблена спеціально для WEB-розробки. Скрипти PHP виконуються на сервері, а результат надсилається клієнту у вигляді звичайного HTML [24].

Java − це високорівнева, об'єктно-орієнтована мова програмування на основі класів, розроблена так, щоб мати якомога менше залежностей при реалізації. Java широко використовується для створення додатків масштабу підприємства, мобільних додатків (особливо для Android), веб-додатків тощо [25].

C# − це сучасна об'єктно-орієнтована мова програмування, розроблена компанією Microsoft як частина її ініціативи .NET. C# призначена бути простою, сучасною, універсальною та безпечною щодо типів мовою програмування [26].

Кожна мова має свої сильні сторони і підходить для різних контекстів, залежно від середовища розробки та вимог проекту. Python є найпростішим у налаштуванні та використанні. Perl - є найменш поширена мовою для Selenium, з більш складним синтаксисом і меншою кількістю ресурсів. В той час як PHP має менш розвинену підтримку для Selenium та краще підходить для завдань веб-розробки. Java є потужною мова з сильною підтримкою спільноти, але більш складна та багатослівна. А C# має хорошу інтеграцію з інструментами .NET, помірну складність та потужну підтримку.

У таблиці 3.2 подано порівняння мов програмування Python, Perl, PHP, Java і C# для автоматизованого тестування в Selenium. Для більшості сценаріїв автоматизованого тестування за допомогою Selenium найкраще підходять Python

та Java завдяки їх широкій підтримці, ресурсам спільноти та тісній прив'язці до Selenium. C# також є гарним вибором, особливо в екосистемі .NET.

Python:

- Переваги: легко вивчати та писати, велика підтримка спільноти, широка підтримка бібліотек та добре підтримувані зв'язки з Selenium.
- Недоліки: повільніша продуктивність порівняно з Java та C#, але для більшості сценаріїв тестування це часто несуттєво.
- Для кого підходить: для початківців і досвідчених користувачів, швидкої розробки скриптів і проектів, де простота використання є пріоритетом.

Таблиця 3.2 – Порівняння мов програмування

Назва критерію	**Python**	**Perl**	**PHP**
Простота використання	Простий у вивченні та написанні	Більш складний і рідше використовується	Рідше використовується для Selenium, веборієнтований
Наявність бібліотек	Великий обсяг бібліотек, з багатьма прикладами та підручниками	Обмежена підтримка бібліотеки Selenium	Обмежена підтримка бібліотеки Selenium
Інтеграція з Selenium	Відмінна	Обмежена	Обмежена
Продуктивність	Загалом добре підходить для завдань автоматизації	Добра, але рідше використовується для цієї мети	Добра, але рідше використовується для автоматизації
Підтримка IDE	Сильна підтримка в популярних IDE (PyCharm, VSCode)	Обмежена підтримка в сучасних IDE	Обмежена підтримка в сучасних IDE
Сумісність з іншими платформами	Міжплатформен-на	Міжплатформенна	Міжплатформенна

Продоження таблиці 3.2

Назва критерію	Java	C#
Простота використання	Помірна складність, широко використовується з Selenium	Помірна складність, хороша інтеграція з .NET
Наявність бібліотек	Великий обсяг бібліотек, з багатьма прикладами та підручниками	Великий обсяг бібліотек, з хорошою інтеграцією з .NET
Інтеграція з Selenium	Відмінна	Відмінна
Продуктивність	Висока продуктивність, особливо у великих проектах	Висока продуктиність, особливо у великих проектах
Підтримка IDE	Сильна підтримка в популярних IDE (IntelliJ, Eclipse)	Сильна підтримка в популярних IDE (Visual Studio)
Сумісність з іншими платформами	Міжплатформенна	Переважно Windows

Java:

- Переваги: висока продуктивність, велика спільнота, велика документація та потужна підтримка бібліотек для Selenium.
- Недоліки: складніша крива навчання, більш багатослівний синтаксис.
- Найкраще для: масштабні проекти, корпоративні додатки та команди з досвідом роботи з Java.

Оцінивши основні відмінності між мовами програмування платформи Selenium, для розробки обрано мову програмування Python. Ця мова програмування легка та зрозуміла у використанні, а також задовольняє основні важливі аспекти, необхідні для розробки автоматизованих тестів, а її недоліки є менш пріоритетними критеріями відносно недоліків аналогів.

Отже, для написання програмного модуля тестування WEB-застосунків буде використана мова програмування Python.

3.3 Опис програмних рішень

Для реалізації автоматизованих тестів мовою Python за допомогою Selenium WebDriver, перш за все потрібно завантажити потрібні драйвери для кожного з браузерів, завантажити програму PyCharm та завантажити сам Python. Усі файли драйверів потрібно розархівувати для подальшої роботи з ними. Під час налаштувань PyCharm обираємо потрібний файл з Python в ролі інтерпретатора та завантажуємо пакет під назвою Selenium. Для того, щоб почати роботу перш за все створюємо новий проект, а в ньому файл з назвою, наприклад «Login test».

Для розширення функціональних можливостей імпортуємо вебдрайвер з пакету Selenium за допомогою такого рядка в коді:

```
from selenium import webdriver
```

Наступним кроком потрібно також імпортувати потрібні біліотеки та функції:

```
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.action_chains import ActionChains
from selenium.webdriver.support.wait import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import time
```

Ці рядки в коді є обов’язковими для кожного автотесту.

Потім створюємо об’єкт serv_obj, в який підключаємо потрібний файл з драйвером для відповідного браузера та відкриваємо браузер. Нижче буде представлений частина коду для браузера Google Chrome:

```
serv_obj = Service("C:/Users/amina/Downloads/БДР/chromedriver-win64/chromedriver.exe")
```

driver = webdriver.Chrome(service=serv_obj)

За допомогою команди driver.get() відкривається потрібне посилання на сайті [27].

Основними командами в коду будуть:

- driver.find_element(By.ID, "") – використовуємо для того, щоб знайти елемент на сторінці за допомогою Id елементу;
- driver.maximize_window() – використовуємо, щоб відкрити сайт на весь екран;
- driver.find_element(By.XPATH, "") – використовуємо для того, щоб знайти елемент на сторінці за допомогою XPath елементу;
- .click() – використовуємо для того, щоб натиснути на елемент;
- .send_keys("") – використовуємо для того, щоб ввести дані в поле елементу;
- time.sleep() – використовуємо для того, щоб зробити паузу;
- driver.title – використовує для того, щоб зафіксувати актуальний заголовок сторінки;
- driver.execute_script("arguments[0].scrollIntoView();",element) – використовуємо для того або пролистати вниз до елементу;
- WebDriverWait(driver,10).until(EC.text_to_be_present_in_element (By.XPATH, "sourse"), "text") – використовуємо для перевірки чи наявний певний текст в елементі;
- print(element.is_selected()) – використовуємо для перевірки чи певний елемент є обраним, в результаті отримуємо значення True або False;
- driver.quit() – використовуємо для завершення роботи програми.

Опираючись на всі ці команди можемо створити автоматизований тест, наприклад для перевірки входу користувача на сайт. Потрібно зробити такі дії:

- зайти на сайт;
- натиснути на кнопку вхід у правому верхньому кутку;
- у спливаючому вікні ввести спершу логін, потім пароль;

- натиснути кнопку «Увійти».

Якщо було введено правильні дані у поля: логін та пароль, користувач буде залогінений у свій акаунт і тест завершиться з вихідним повідомленням «Login Test Passed», тобто тест пройшов успішно. Якщо ж інформація була введена неправильно, буде виведений результат «Login Test Failed». Нижче буде подано код одного з розроблених автоматизованих тестів.

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.chrome.service import Service
import time
serv_obj = Service("C:/Users/amina/Downloads/БДР/chromedriver-
win64/chromedriver.exe")
driver = webdriver.Chrome(service=serv_obj)
driver.get("https://nashformat.ua/")
time.sleep(5)
enter = driver.find_element(By.ID, "user-menu")
enter.click()
time.sleep(2)
driver.find_element(By.ID, "loginPopup")
email = driver.find_element(By.XPATH, "//*[@id='formLogin']/div[2]/input")
email.click()
email.send_keys("Буквар")
time.sleep(5)
password = driver.find_element(By.XPATH,
"//*[@id='formLogin']/div[3]/input")
password.click()
time.sleep(2)
```

```
password.send_keys("Буквар")
time.sleep(5)
enter = driver.find_element(By.XPATH, "//*[@id='formLogin']/div[4]/input[2]")
enter.click()
time.sleep(5)
act_title = driver.title
exp_title = "Мій аккаунт"
if act_title == exp_title:
    print("Login Test Passed")
else:
   print("Login Test Failed")
driver.quit()
```

3.4 Тестування роботи програми та аналіз результатів

Тестування програмного забезпечення є ключовим етапом у процесі його розробки. Його метою є перевірка відповідності програми встановленим вимогам, забезпечення її бездоганного і надійного функціонування. Одне з завдань тестування полягає у виявленні помилок, недоліків і проблем, що можуть виникати під час роботи програми.

Було проведено 50 тестових запусків кожного скрипта з різними тестовими даними. Перш з все роглянемо роботу скрипту для входу в акаунт користувача. Для тестування нам потрібно перш за все ввести тестові дані: адресу електронної пошти та пароль. Наприклад, розглянемо два випадки з дійсним логіном та паролем(amina.melnychuk@gmail.com, *********), а також з хибними даними.

При запуску скрипту з валідними даними бачимо, що вводяться дані в поля (рис. 3.1) та потім отримаємо результат, зображений на рисунку 3.2. Також нижче на рисинках 3.3 та 3.4 буде представлені результати роботи програми з неправильними даними, наприклад, якщо у всі поля ввести "Буквар".

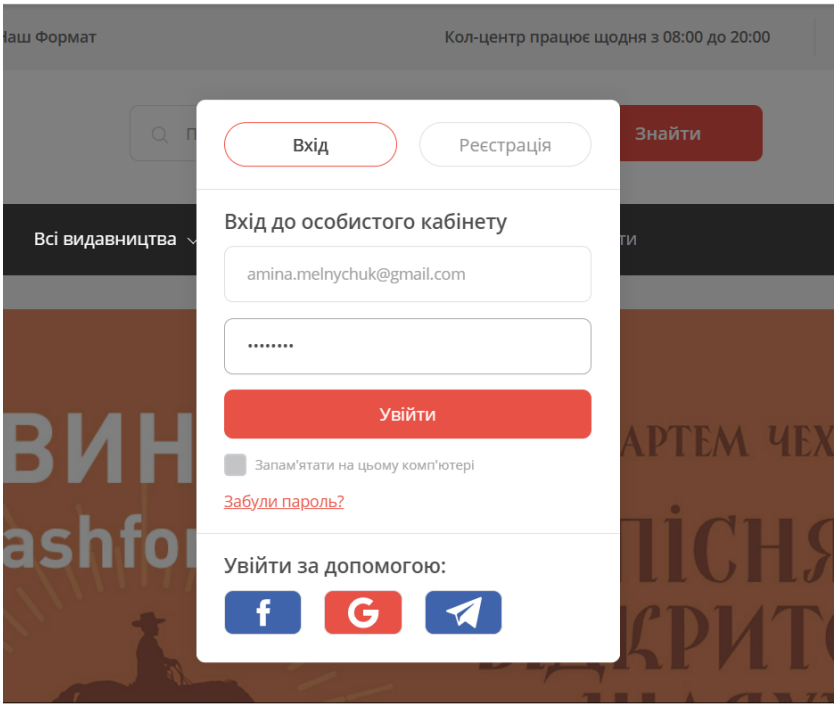


Рисунок 3.1 – Вікно введення даних з валідними значеннями

```
driver.find_element(By.ID, value: "loginPopup")

email = driver.find_element(By.XPATH, value: "//*[@id='formLogin']/div[2]/input")
email.click()
email.send_keys("amina.melnychuk@gmail.com")
time.sleep(5)

password = driver.find_element(By.XPATH, value: "//*[@id='formLogin']/div[3]/input")
password.click()
time.sleep(2)
password.send_keys("Dk10gh@m")
time.sleep(5)

enter = driver.find_element(By.XPATH, value: "//*[@id='formLogin']/div[4]/input[2]")
```

```
Login Test Chrome
C:\Users\amina\AppData\Local\Programs\Python\Python312\python.exe "C:\Users\amina\Downloads\БДР\Login Test Chrome.py"
Login Test Passed

Process finished with exit code 0
```

Рисунок 3.2 – Вікно з отриманим результатом "Login Test Passed"

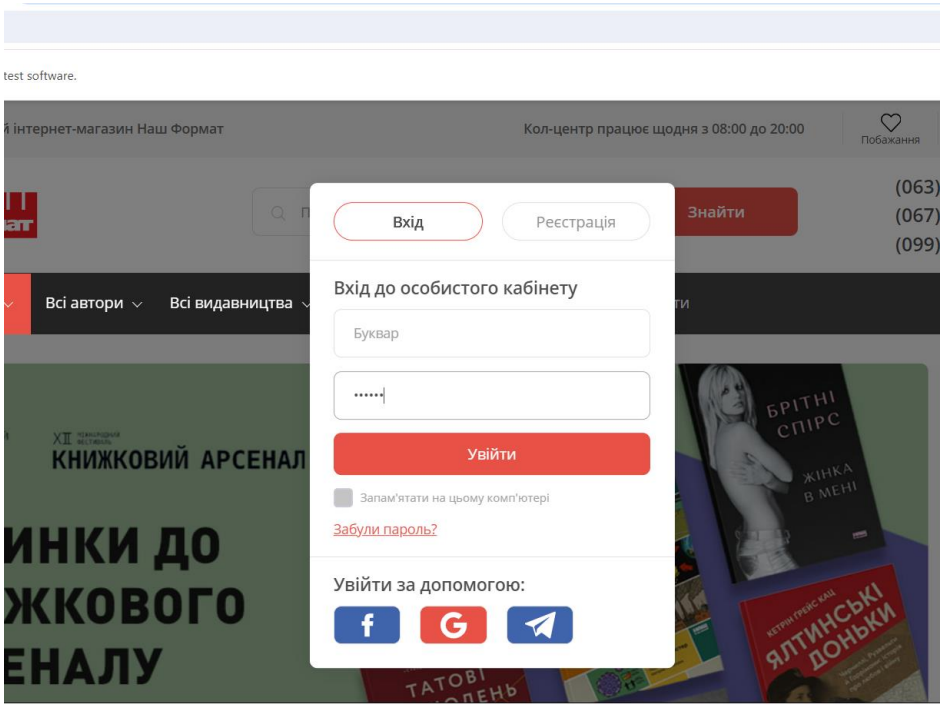


Рисунок 3.3 – Вікно введення даних з неправильними значеннями

Рисунок 3.4 – Вікно з отриманим результатом "Login Test Failed"

У процесі тестування роботи створених автоматизованих тестів було виявлено ще одну помилку в роботі сайту. Її добре видно при запуску скрипта для додавання книги в уподобання. Тож розлянемо тестування цього автоматизованого тесту. Перш за все користувач має зайти в свій акаунт, натиснути на пошук та ввести, наприклад, "Буквар"(рис. 3.5).

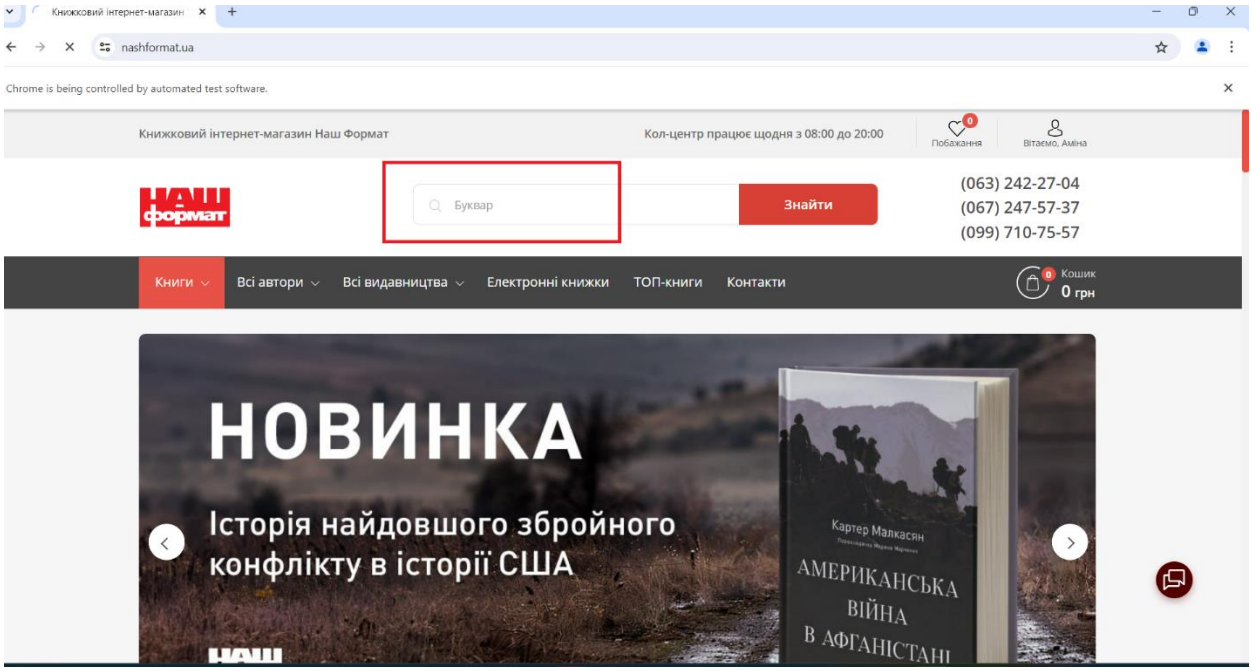


Рисунок 3.5 – Вікно введення даних в поле Пошук

Потім користувач обирає книгу та натискає на сердечко. В спливаючому вікні вводить назву нової полиці та натискає Створити та помістити (рис. 3.6).

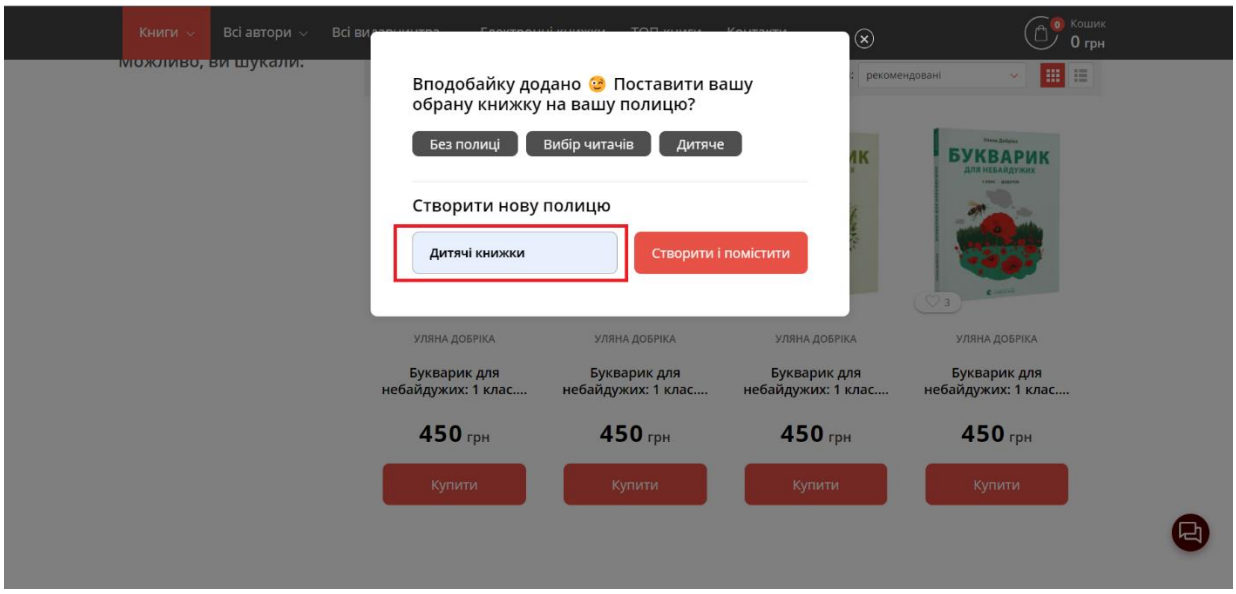


Рисунок 3.6 – Вікно створення нової полиці для обраної книги

Очікуваним результатом таких дій користувача, книга має бути додана в уподобання і має горіти цифра 1 в правому верхньому кутку. А натомість показник

не змінюється з 0 на 1 та кнопка для переходу на сторінку вподобань перестає бути активною (рис. 3.7).

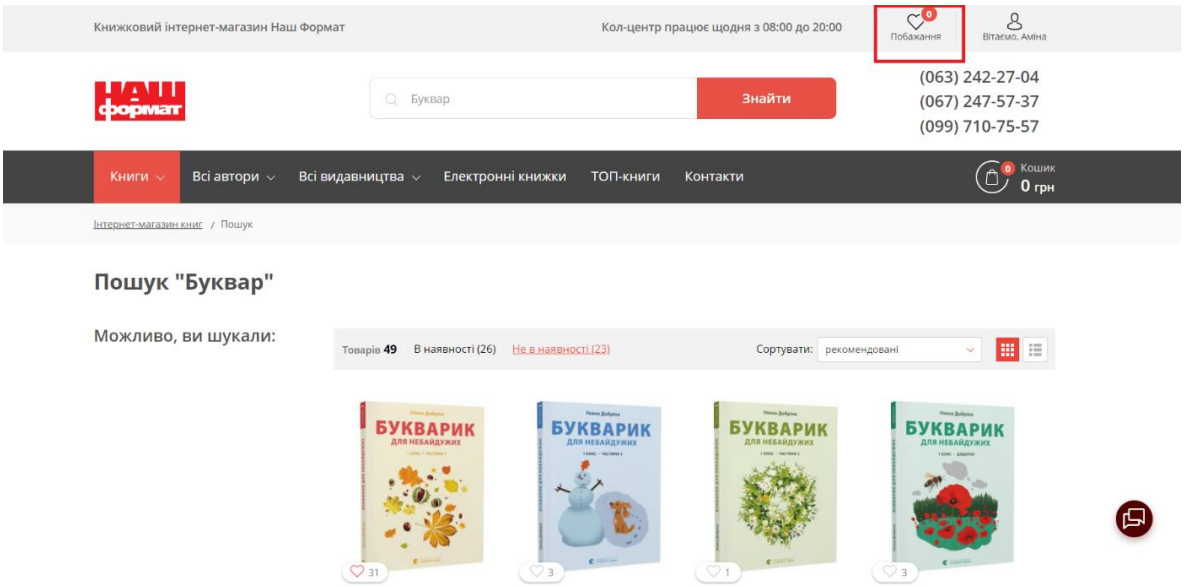


Рисунок 3.7 – Показник кількості вподобаних книг

Отже, в результаті тест нам видає повідомлення False, яке означає, що кнопка переходу до вподобань не активна. А також видає повідомлення про те, що тест не пройшов успішно, адже не змінився показник вподобаних книг. Це представлено на рисунку 3.8.

Рисунок 3.8 – Вікно з отриманим результатом "Add to wishes Test Failed"

Розроблений програмний модуль тестування WEB-застосунків був протестований спеціалістами в галузі тестування програмного забезпечення за такими критеріями:

- структурованість;
- чіткість та однозначність;
- володіння матеріалом;
- ефективність;
- доцільність.

Для тестування на достовірність правдивості відповідей було обрано десять спеціалістів, які працюють в галузі QA. Оцінка роботи виставлялась від 1 до 10, в залежності від того розробка відповідала критеріям чи ні. Результати тестування наведені в таблиці 3.3.

Таблиця 3.3 - Результати тестування створеного програмного модуля

Спеціаліст	1	2	3	4	5	6	7	8	9	10
Оцінка	10	8	8	7	8	9	8	7	9	10

Проаналізувавши дані, маємо загальну оцінку 84 із 100. Отже, можна дійти висновку, що якість роботи нового програмного продукту складає 84%. Це свідчить про досить високий рівень підвищення якості роботи програмного модуля тестування WEB-застосунків.

3.5 Висновок

У результаті проведеного аналізу розробленого програмного модуля автоматичного тестування WEB-застосунку можна стверджувати, що поставлені цілі були досягнуті. Розроблені автоматичні тести показують, як можна ефективніше та швидше проводити тестування, за допомогою перенесення певної частини функціоналу на автоматичне тестування.

Проведений аналіз мов програмування, які можуть бути використані для написання автоматичних тестів, дав можливість обрати найоптимальніший

варіант. Обраний модуть є зручним та зрозумілим для написання скриптів, має зручний інтерфейс та є інтегрованим з іншими модулями, які розширюють можливості автоматичних тестів в покритті тестового функціоналу.

Розроблені тести можуть знайти місце на будь-якому WEB-ресурсі, адже вони покривають стандартні дії користувача з сайтом. Такі автоматизовані тести можуть бути використані в усіх браузерах за наявності завантажених драйверів, їх роботу було представлено на основі браузерів Google Chrome та Microsoft Edge.

Розроблений програмний модуль тестування WEB-застосунків був успішно протестований спеціалістами в галузі тестування програмного забезпечення. Якість роботи нового програмного продукту складає 84%.

Таким чином, результати аналізу та тестування підтверджують успішну реалізацію поставлених цілей.

ВИСНОВКИ

Всі задачі, поставлені для реалізації програмного модуля тестування WEB-застосунків були виконані в повному обсязі, а саме: обґрунтувана доцільність розробки програмного модуля тестування WEB-застосунків; розроблено програмний модуль тестування WEB-застосунків; програмно реалізувано модуль тестування WEB-застосунків; виконано тестування програми та проведено аналіз отриманих результатів; розроблено інструкцію користувача.

Для розробки автоматичних тестів використана програма Pycharm та розширення Selenium WebDriver в поєднання з мовою програмування Python. Платформами для реалізації було обрано браузери Google Chrome та Microsoft Edge.

Для реалізації були розроблені спершу користувацькі сценарії, вимоги, чек-лист, тест-кейси, баг-репорти, а також були розроблені алгоритми для покриття автоматизованими тестами WEB-застосунку.

Розроблений програмний модуль тестування WEB-застосунків був успішно протестований спеціалістами в галузі тестування програмного забезпечення. В результаті тестування було виявлено критичний дефект в роботі ресурсу, тож можна вважати, що розроблені автоматизовані тести виконують своє основне завдання і реалізація програмного модуля тестування є успішною.

Мета роботи, яка полягала в збільшення якості роботи програмного забезпечення автоматизованого тестування WEB-застосунків, досягнута за рахунок детального вивчення процесу створення тестових артефактів, опрацювання можливих платформ, мов програмування для реалізації та розробки самих автоматизованих тестів таким чином, що вони покривають базові дії користувача на сайті. Якість роботи нового програмного продукту складає 84%.

Отже, розроблений програмний модуль тестування WEB-застосунків відповідає заданим вимогам.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Мельничук А. М., Крилик Л.В. Аналіз передумов розробки програмного модуля тестування ВЕБ-застосунків. *Міжнародна науково-практична інтернет-конференція студентів аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи (МН-2024), м. Вінниці у 2024 р.* URL: https://conferences.vntu.edu.ua/index.php/mn/mn2024/paper/view/21059 (дата звернення: 01.06.2024).

2. Що таке тестування ПЗ, його етапи, види, інструменти – Sigma Software University. URL: https://university.sigma.software/what-is-software-testing/ (дата звернення: 01.06.2024).

3. SDLC - Overview – tutorialspoint. URL: https://www.tutorialspoint.com/sdlc/sdlc_overview.htm (дата звернення: 01.06.2024).

4. SDLC - Waterfall Model – tutorialspoint. URL: https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm (дата звернення: 01.06.2024).

5. SDLC - V-Model – tutorialspoint. URL: https://www.tutorialspoint.com/sdlc/sdlc_v_model.htm (дата звернення: 01.06.2024).

6. SDLC - Agile Model – tutorialspoint. URL: https://www.tutorialspoint.com/sdlc/sdlc_agile_model.htm (дата звернення: 01.06.2024).

7. Види тестування та відмінності між ними – QualityAssuranceGroup. URL: https://qagroup.com.ua/publications/vydy-testuvannya-ta-vidminnosti-mizh-nymy/ (дата звернення: 01.06.2024).

8. White/black/grey box-тестування – QALight. URL: https://qalight.ua/baza-znaniy/white-black-grey-box-testuvannya/ (дата звернення: 01.06.2024).

9. Software Testing - Methods – tutorialspoint. URL: https://www.tutorialspoint.com/software_testing/software_testing_methods.htm (дата звернення: 01.06.2024).

10. Differences Between Verification and Validation – GURU99. URL: https://www.guru99.com/verification-v-s-validation-in-a-software-testing.html (дата звернення: 01.06.2024).

11. TEST PLAN in Software Testing (Example) – GURU99. URL: https://www.guru99.com/test-planning.html (дата звернення: 01.06.2024).

12. Test cases – Google Docs. URL: https://docs.google.com/spreadsheets/d/1HWvbWXLLahLz6HRROFGa0lg44dplTGSyx1Hl6hhoxdE/edit?usp=sharing (дата звернення: 01.06.2024).

12. Introduction of Test Artifacts – Geeks for Geeks. URL: https://www.geeksforgeeks.org/introduction-of-test-artifacts/ (дата звернення: 01.06.2024).

13. User Stories In Testing: How To Convert it Into Test Cases? – testsigma. URL: https://testsigma.com/blog/user-stories-in-testing/ (дата звернення: 01.06.2024).

14. User Story – CenterCode. URL: https://www.centercode.com/glossary/user-story (дата звернення: 01.06.2024).

15. Importance of Software Requirements Specification (SRS) – Geeks for Geeks. URL: https://www.geeksforgeeks.org/importance-of-software-requirements-specification-srs/ (дата звернення: 01.06.2024).

16. Testing CheckLists – QATestLab. URL: https://qatestlab.com/resources/knowledge-center/sample-deliverables/testing-checklists/ (дата звернення: 01.06.2024).

17. How to write an Effective Bug Report – BrowserStack. URL: https://www.browserstack.com/guide/how-to-write-a-bug-report (дата звернення: 01.06.2024).

18. Webdriver – Selenium. URL: https://www.selenium.dev/documentation/webdriver/ (дата звернення: 01.06.2024).

19. What is Postman? – Postman. URL: https://www.postman.com/ (дата звернення: 01.06.2024).

20. JMeter Test Automation – Testable. URL: https://testable.io/jmeter?gad_source=1&gclid=CjwKCAjwyJqzBhBaEiwAWDRJVFbJ_uBySGVPWgMutd9tlQalYd5YnopvLdGpRkt2sE6K1_wzR3yknxoCg94QAvD_BwE (дата звернення: 01.06.2024).

21. Що таке TestNG? − QualityAssuranceGroup. URL: https://qagroup.com.ua/publications/what-is-testng/ (дата звернення: 01.06.2024).

22. Python – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Python (дата звернення: 01.06.2024).

23. Perl – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Perl (дата звернення: 01.06.2024).

24. PHP – Вікіпедія. URL: https://uk.wikipedia.org/wiki/PHP (дата звернення: 01.06.2024).

25. Java – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Java (дата звернення: 01.06.2024).

26. C# – Вікіпедія. URL: https://uk.wikipedia.org/wiki/C_Sharp (дата звернення: 01.06.2024).

27. Наш Формат – Книжковий інтернет-магазин. URL: https://nashformat.ua/ (дата звернення: 01.06.2024).

ДОДАТКИ

ДОДАТОК А (обов'язковий)

Протокол перевірки кваліфікаційної роботи на наявність текстових запозичень

Назва роботи: Програмний модуль тестування WEB-застосунків

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ кафедра комп'ютерних наук, ФІТА
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 95,34% Схожість 4,66%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку Озеранський В.С.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи Мельничук А.М.

Керівник роботи Крилик Л.В.

ДОДАТОК Б (обов'язковий)

Лістинг програми

1. Login Test Chrome:

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.support.wait import WebDriverWait
import time
from selenium.webdriver.support import expected_conditions as EC

serv_obj = Service("C:/Users/amina/Downloads/БДР/chromedriver-win64/chromedriver.exe")
driver = webdriver.Chrome(service=serv_obj)

driver.get("https://nashformat.ua/")
driver.maximize_window()
time.sleep(5)
enter = driver.find_element(By.ID, "user-menu")
enter.click()
time.sleep(2)
driver.find_element(By.ID, "loginPopup")

email = driver.find_element(By.XPATH, "//*[@id='formLogin']/div[2]/input")
email.click()
email.send_keys("amina.melnychuk@gmail.com")
time.sleep(5)

password = driver.find_element(By.XPATH, "//*[@id='formLogin']/div[3]/input")
```

```
password.click()
time.sleep(2)
password.send_keys("Dk10gh@m")
time.sleep(5)

enter = driver.find_element(By.XPATH, "//*[@id='formLogin']/div[4]/input[2]")
enter.click()
time.sleep(5)

if WebDriverWait(driver, 10).until(EC.text_to_be_present_in_element((By.XPATH,
"//*[@id='user-menu']/a"), 'Вітаємо, Аміна')):
   print("Login Test Passed")
else:
   print("Login Test Failed")

driver.quit()
```

2. Login Test Edge:

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.chrome.service import Service
import time

serv_obj = Service("C:/Users/amina/Downloads/БДР/edge/msedgedriver.exe")
driver = webdriver.Edge(service=serv_obj)

driver.get("https://nashformat.ua/")
driver.maximize_window()
time.sleep(5)
```

```
enter = driver.find_element(By.ID, "user-menu")
enter.click()
time.sleep(2)
driver.find_element(By.ID, "loginPopup")

email = driver.find_element(By.XPATH, "//*[@id='formLogin']/div[2]/input")
email.click()
email.send_keys("Буквар")
time.sleep(5)

password = driver.find_element(By.XPATH, "//*[@id='formLogin']/div[3]/input")
password.click()
time.sleep(2)
password.send_keys("Буквар")
time.sleep(5)

enter = driver.find_element(By.XPATH, "//*[@id='formLogin']/div[4]/input[2]")
enter.click()
time.sleep(5)

act_title = driver.title
exp_title = "Мій аккаунт"

if act_title == exp_title:
    print("Login Test Passed")
else:
    print("Login Test Failed")

driver.quit()
```

3. Add delivery address Test Chrome:

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.support.wait import WebDriverWait
import time
from selenium.webdriver.support import expected_conditions as EC

serv_obj = Service("C:/Users/amina/Downloads/БДР/chromedriver-win64/chromedriver.exe")
driver = webdriver.Chrome(service=serv_obj)

driver.get("https://nashformat.ua/")
driver.maximize_window()
time.sleep(3)

enter = driver.find_element(By.ID, "user-menu")
enter.click()
time.sleep(2)
driver.find_element(By.ID, "loginPopup")

email = driver.find_element(By.XPATH, "//*[@id='formLogin']/div[2]/input")
email.click()
email.send_keys("amina.melnychuk@gmail.com")
time.sleep(3)

password = driver.find_element(By.XPATH, "//*[@id='formLogin']/div[3]/input")
password.click()
time.sleep(2)
password.send_keys("Dk10gh@m")
```

```
time.sleep(3)

enter = driver.find_element(By.XPATH, "//*[@id='formLogin']/div[4]/input[2]")
enter.click()
time.sleep(3)

account = driver.find_element(By.XPATH, "//*[@id='user-menu']/a")
account.click()
time.sleep(2)

drop_list = driver.find_element(By.XPATH, "//*[@id='user-menu']/div")
time.sleep(3)

my_acc = driver.find_element(By.XPATH, "//*[@id='user-menu']/div/div/ul/li[1]")
time.sleep(3)
my_acc.click()

delivery = driver.find_element(By.XPATH, "//*[@id='fn-content']/div[2]/div/div[1]/div/ul/li[4]/a")
time.sleep(2)
delivery.click()

new_address = driver.find_element(By.XPATH, "//*[@id='fn-content']/div[2]/div/div[2]/div[1]/a")
time.sleep(2)
new_address.click()
time.sleep(2)

new_address_popup = driver.find_element(By.XPATH, "//*[@id='add-delivery']/div")
time.sleep(2)
```

```
post = driver.find_element(By.XPATH, "//*[@id='delivery_4']/label")
post.click()

city = driver.find_element(By.XPATH, "//*[@id='select2-npcity-container']")
driver.execute_script("arguments[0].scrollIntoView();", city)
time.sleep(2)
city.click()
search_city = driver.find_element(By.XPATH, "/html/body/span/span/span[1]/input")
time.sleep(1)
search_city.click()
time.sleep(1)
search_city.send_keys("Вінниця")
time.sleep(2)
option_city = driver.find_element(By.XPATH, "//*[@id='select2-npcity-results']/li")
time.sleep(1)
option_city.click()
time.sleep(2)

office = driver.find_element(By.XPATH,
"//*[@id='delivery_new_post']/div[4]/span/span[1]/span")
driver.execute_script("arguments[0].scrollIntoView();", office)
time.sleep(2)
office.click()
#search_dropdown = driver.find_element(By.XPATH, "/html/body/span/span/span[1]")
#time.sleep(1)
search_office = driver.find_element(By.XPATH, "/html/body/span/span/span[1]/input")
time.sleep(1)
search_office.click()
time.sleep(1)
```

```
search_office.send_keys("12")
time.sleep(2)
option_office = driver.find_element(By.XPATH, "//*[@id='select2-npware-results']/li[1]/span")
time.sleep(1)
option_office.click()
time.sleep(2)

save_button = driver.find_element(By.XPATH, "//*[@id='fn-save_user_delivery']")
time.sleep(1)
save_button.click()
time.sleep(5)

act_result = driver.find_element(By.XPATH, "//*[@id='sortable']/tbody/tr[2]/td[3]/div[2]")
#driver.execute_script("arguments[0].scrollIntoView();", act_result)
time.sleep(5)

#exp_result = "Вінниця, Відділення №12 (до 30 кг на одне місце ): вул. Пирогова, 31"
#result = WebDriverWait(driver, 10).until(EC.text_to_be_present_in_element((By.XPATH, "//*[@id='sortable']/tbody/tr[2]/td[3]/div[2]"), 'Вінниця, Відділення №12 (до 30 кг на одне місце ): вул. Пирогова, 31'))

if WebDriverWait(driver, 10).until(EC.text_to_be_present_in_element((By.XPATH, "//*[@id='sortable']/tbody/tr[2]/td[3]/div[2]"), 'Вінниця, Відділення №12 (до 30 кг на одне місце ): вул. Пирогова, 31')):
    print("Delivery address was successfully added. Test Passed")
else:
```

```
    print("Delivery address was not added. Test Failed")

driver.quit()
```

4. Change delivery address Test Chrome:

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.support.wait import WebDriverWait
import time
from selenium.webdriver.support import expected_conditions as EC

serv_obj = Service("C:/Users/amina/Downloads/БДР/chromedriver-win64/chromedriver.exe")
driver = webdriver.Chrome(service=serv_obj)

driver.get("https://nashformat.ua/")
driver.maximize_window()
time.sleep(3)

enter = driver.find_element(By.ID, "user-menu")
enter.click()
time.sleep(2)
driver.find_element(By.ID, "loginPopup")

email = driver.find_element(By.XPATH, "//*[@id='formLogin']/div[2]/input")
email.click()
email.send_keys("amina.melnychuk@gmail.com")
time.sleep(3)
```

```
password = driver.find_element(By.XPATH, "//*[@id='formLogin']/div[3]/input")
password.click()
time.sleep(2)
password.send_keys("Dk10gh@m")
time.sleep(3)

enter = driver.find_element(By.XPATH, "//*[@id='formLogin']/div[4]/input[2]")
enter.click()
time.sleep(3)

account = driver.find_element(By.XPATH, "//*[@id='user-menu']/a")
account.click()
time.sleep(2)

drop_list = driver.find_element(By.XPATH, "//*[@id='user-menu']/div")
time.sleep(3)

my_acc = driver.find_element(By.XPATH, "//*[@id='user-menu']/div/div/ul/li[1]")
time.sleep(3)
my_acc.click()

delivery = driver.find_element(By.XPATH, "//*[@id='fn-content']/div[2]/div/div[1]/div/ul/li[4]/a")
time.sleep(2)
delivery.click()

modify = driver.find_element(By.XPATH, "//*[@id='sortable']/tbody/tr[1]/td[5]/div[3]/a")
modify.click()
```

```
modify_popup = driver.find_element(By.XPATH, "//*[@id='add-delivery']/div")
time.sleep(2)

city = driver.find_element(By.XPATH, "//*[@id='select2-npcity-container']")
driver.execute_script("arguments[0].scrollIntoView();", city)
time.sleep(2)
city.click()
search_city = driver.find_element(By.XPATH, "/html/body/span/span/span[1]/input")
time.sleep(1)
search_city.click()
time.sleep(1)
search_city.send_keys("Агрономічне")
time.sleep(2)
option_city = driver.find_element(By.XPATH, "//*[@id='select2-npcity-results']/li[1]")
time.sleep(1)
option_city.click()
time.sleep(2)

office = driver.find_element(By.XPATH,
"//*[@id='delivery_new_post']/div[4]/span/span[1]/span")
driver.execute_script("arguments[0].scrollIntoView();", office)
time.sleep(2)
office.click()
#search_dropdown = driver.find_element(By.XPATH, "/html/body/span/span/span[1]")
#time.sleep(1)
search_office = driver.find_element(By.XPATH, "/html/body/span/span/span[1]/input")
time.sleep(1)
search_office.click()
time.sleep(1)
search_office.send_keys("1")
```

```
time.sleep(2)
option_office = driver.find_element(By.XPATH, "//*[@id='select2-npware-results']/li[1]")
time.sleep(1)
option_office.click()
time.sleep(2)

save_button = driver.find_element(By.XPATH, "//*[@id='fn-save_user_delivery']")
time.sleep(1)
save_button.click()
time.sleep(5)

act_result = driver.find_element(By.XPATH, "//*[@id='sortable']/tbody/tr[1]")
#driver.execute_script("arguments[0].scrollIntoView();", act_result)
time.sleep(5)

if WebDriverWait(driver, 10).until(EC.text_to_be_present_in_element((By.XPATH, "//*[@id='sortable']/tbody/tr[1]/td[3]/div[2]"), 'Агрономічне, Відділення №1: вул. Мічуріна, 2а')):
    print("Delivery address was successfully changed. Test Passed")
else:
    print("Delivery address was not changed. Test Failed")

driver.quit()
```

5. Add delivery address Test Edge:

```
import time
from selenium.webdriver.support import expected_conditions as EC

from selenium import webdriver
```

```
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.support.wait import WebDriverWait

serv_obj = Service("C:/Users/amina/Downloads/БДР/edge/msedgedriver.exe")
driver = webdriver.Edge(service=serv_obj)

driver.get("https://nashformat.ua/")
driver.maximize_window()
time.sleep(3)

enter = driver.find_element(By.ID, "user-menu")
enter.click()
time.sleep(2)
driver.find_element(By.ID, "loginPopup")

email = driver.find_element(By.XPATH, "//*[@id='formLogin']/div[2]/input")
email.click()
email.send_keys("amina.melnychuk@gmail.com")
time.sleep(3)

password = driver.find_element(By.XPATH, "//*[@id='formLogin']/div[3]/input")
password.click()
time.sleep(2)
password.send_keys("Dk10gh@m")
time.sleep(3)

enter = driver.find_element(By.XPATH, "//*[@id='formLogin']/div[4]/input[2]")
enter.click()
time.sleep(3)
```

```
account = driver.find_element(By.XPATH, "//*[@id='user-menu']/a")
account.click()
time.sleep(2)

drop_list = driver.find_element(By.XPATH, "//*[@id='user-menu']/div")
time.sleep(3)

my_acc = driver.find_element(By.XPATH, "//*[@id='user-menu']/div/div/ul/li[1]")
time.sleep(3)
my_acc.click()

delivery = driver.find_element(By.XPATH, "//*[@id='fn-content']/div[2]/div/div[1]/div/ul/li[4]/a")
time.sleep(2)
delivery.click()

new_address = driver.find_element(By.XPATH, "//*[@id='fn-content']/div[2]/div/div[2]/div[1]/a")
time.sleep(2)
new_address.click()
time.sleep(2)

new_address_popup = driver.find_element(By.XPATH, "//*[@id='add-delivery']/div")
time.sleep(2)

post = driver.find_element(By.XPATH, "//*[@id='delivery_4']/label")
post.click()

city = driver.find_element(By.XPATH, "//*[@id='select2-npcity-container']")
```

```
driver.execute_script("arguments[0].scrollIntoView();", city)
time.sleep(2)
city.click()
search_city = driver.find_element(By.XPATH, "/html/body/span/span/span[1]/input")
time.sleep(1)
search_city.click()
time.sleep(1)
search_city.send_keys("Вінниця")
time.sleep(2)
option_city = driver.find_element(By.XPATH, "//*[@id='select2-npcity-results']/li")
time.sleep(1)
option_city.click()
time.sleep(2)

office = driver.find_element(By.XPATH, "//*[@id='delivery_new_post']/div[4]/span/span[1]/span")
driver.execute_script("arguments[0].scrollIntoView();", office)
time.sleep(2)
office.click()
#search_dropdown = driver.find_element(By.XPATH, "/html/body/span/span/span[1]")
#time.sleep(1)
search_office = driver.find_element(By.XPATH, "/html/body/span/span/span[1]/input")
time.sleep(1)
search_office.click()
time.sleep(1)
search_office.send_keys("12")
time.sleep(2)
option_office = driver.find_element(By.XPATH, "//*[@id='select2-npware-results']/li[1]/span")
time.sleep(1)
```

```
option_office.click()
time.sleep(2)

save_button = driver.find_element(By.XPATH, "//*[@id='fn-save_user_delivery']")
time.sleep(1)
save_button.click()
time.sleep(5)

act_result = driver.find_element(By.XPATH, "//*[@id='sortable']/tbody/tr[2]/td[3]/div[2]")
#driver.execute_script("arguments[0].scrollIntoView();", act_result)
time.sleep(5)

#exp_result = "Вінниця, Відділення №12 (до 30 кг на одне місце ): вул. Пирогова, 31"
#result = WebDriverWait(driver, 10).until(EC.text_to_be_present_in_element((By.XPATH, "//*[@id='sortable']/tbody/tr[2]/td[3]/div[2]"), 'Вінниця, Відділення №12 (до 30 кг на одне місце ): вул. Пирогова, 31'))

if WebDriverWait(driver, 10).until(EC.text_to_be_present_in_element((By.XPATH, "//*[@id='sortable']/tbody/tr[2]/td[3]/div[2]"), 'Вінниця, Відділення №12 (до 30 кг на одне місце ): вул. Пирогова, 31')):
    print("Delivery address was successfully added. Test Passed")
else:
    print("Delivery address was not added. Test Failed")

driver.quit()
```

6. Change delivery address Test Edge:

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.support.wait import WebDriverWait
import time
from selenium.webdriver.support import expected_conditions as EC

serv_obj = Service("C:/Users/amina/Downloads/БДР/edge/msedgedriver.exe")
driver = webdriver.Edge(service=serv_obj)

driver.get("https://nashformat.ua/")
driver.maximize_window()
time.sleep(3)

enter = driver.find_element(By.ID, "user-menu")
enter.click()
time.sleep(2)
driver.find_element(By.ID, "loginPopup")

email = driver.find_element(By.XPATH, "//*[@id='formLogin']/div[2]/input")
email.click()
email.send_keys("amina.melnychuk@gmail.com")
time.sleep(3)

password = driver.find_element(By.XPATH, "//*[@id='formLogin']/div[3]/input")
password.click()
time.sleep(2)
password.send_keys("Dk10gh@m")
time.sleep(3)
```

```
enter = driver.find_element(By.XPATH, "//*[@id='formLogin']/div[4]/input[2]")
enter.click()
time.sleep(3)

account = driver.find_element(By.XPATH, "//*[@id='user-menu']/a")
account.click()
time.sleep(2)

drop_list = driver.find_element(By.XPATH, "//*[@id='user-menu']/div")
time.sleep(3)

my_acc = driver.find_element(By.XPATH, "//*[@id='user-menu']/div/div/ul/li[1]")
time.sleep(3)
my_acc.click()

delivery = driver.find_element(By.XPATH, "//*[@id='fn-content']/div[2]/div/div[1]/div/ul/li[4]/a")
time.sleep(2)
delivery.click()

modify = driver.find_element(By.XPATH, "//*[@id='sortable']/tbody/tr[1]/td[5]/div[3]/a")
modify.click()

modify_popup = driver.find_element(By.XPATH, "//*[@id='add-delivery']/div")
time.sleep(2)

city = driver.find_element(By.XPATH, "//*[@id='select2-npcity-container']")
driver.execute_script("arguments[0].scrollIntoView();", city)
time.sleep(2)
```

```
city.click()
search_city = driver.find_element(By.XPATH, "/html/body/span/span/span[1]/input")
time.sleep(1)
search_city.click()
time.sleep(1)
search_city.send_keys("Агрономічне")
time.sleep(2)
option_city = driver.find_element(By.XPATH, "//*[@id='select2-npcity-results']/li[1]")
time.sleep(1)
option_city.click()
time.sleep(2)

office = driver.find_element(By.XPATH, "//*[@id='delivery_new_post']/div[4]/span/span[1]/span")
driver.execute_script("arguments[0].scrollIntoView();", office)
time.sleep(2)
office.click()
#search_dropdown = driver.find_element(By.XPATH, "/html/body/span/span/span[1]")
#time.sleep(1)
search_office = driver.find_element(By.XPATH, "/html/body/span/span/span[1]/input")
time.sleep(1)
search_office.click()
time.sleep(1)
search_office.send_keys("1")
time.sleep(2)
option_office = driver.find_element(By.XPATH, "//*[@id='select2-npware-results']/li[1]")
time.sleep(1)
option_office.click()
time.sleep(2)
```

```
save_button = driver.find_element(By.XPATH, "//*[@id='fn-save_user_delivery']")
time.sleep(1)
save_button.click()
time.sleep(5)

act_result = driver.find_element(By.XPATH, "//*[@id='sortable']/tbody/tr[1]")
#driver.execute_script("arguments[0].scrollIntoView();", act_result)
time.sleep(5)

if WebDriverWait(driver, 10).until(EC.text_to_be_present_in_element((By.XPATH, "//*[@id='sortable']/tbody/tr[1]/td[3]/div[2]"), 'Агрономічне, Відділення №1: вул. Мічуріна, 2а')):
   print("Delivery address was successfully changed. Test Passed")
else:
   print("Delivery address was not changed. Test Failed")

driver.quit()
```

7. Add to wishes Test Chrome:

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.action_chains import ActionChains
import time

serv_obj = Service("C:/Users/amina/Downloads/БДР/chromedriver-win64/chromedriver.exe")
driver = webdriver.Chrome(service=serv_obj)
```

```
driver.get("https://nashformat.ua/")
driver.maximize_window()
time.sleep(1)

enter = driver.find_element(By.ID, "user-menu")
enter.click()
time.sleep(1)
driver.find_element(By.ID, "loginPopup")

email = driver.find_element(By.XPATH, "//*[@id='formLogin']/div[2]/input")
email.click()
email.send_keys("amina.melnychuk@gmail.com")
time.sleep(1)

password = driver.find_element(By.XPATH, "//*[@id='formLogin']/div[3]/input")
password.click()
time.sleep(1)
password.send_keys("Dk10gh@m")
time.sleep(1)

enter = driver.find_element(By.XPATH, "//*[@id='formLogin']/div[4]/input[2]")
enter.click()
time.sleep(1)

driver.find_element(By.XPATH, "//*[@id='searchHeaderPlace']")
search = driver.find_element(By.XPATH, "//*[@id='fn-search']/label/input")
search.click()
search.send_keys("Буквар")
time.sleep(2)
```

```
find_button = driver.find_element(By.XPATH, "//*[@id='fn-search']/span/button")
find_button.click()
time.sleep(5)

heart = driver.find_element(By.XPATH, "//*[@id='fn-products_content']/div[1]/div/div/div[1]/div/a[2]/i")
driver.execute_script("arguments[0].scrollIntoView();", heart)
time.sleep(2)
driver.execute_script("arguments[0].click();", heart)
time.sleep(2)

popup = driver.find_element(By.XPATH, "//*[@id='shelvesPopup']/div/div")

shelf = driver.find_element(By.XPATH, "//*[@id='new_shelf_name']")
#driver.execute_script("arguments[0].scrollIntoView();", close_button)
driver.execute_script("arguments[0].click();", shelf)
time.sleep(5)
shelf.send_keys("Дитячі книжки")
time.sleep(5)
#print(close_button.is_selected())
create_button = driver.find_element(By.XPATH, "//*[@id='shelvesPopup']/div/div/div/div[3]/div[2]/div/form/button")
driver.execute_script("arguments[0].click();", create_button)
time.sleep(5)

wishlist = driver.find_element(By.XPATH, "//*[@id='wishlist']/span")
driver.execute_script("arguments[0].scrollIntoView();", wishlist)
time.sleep(3)
if print(wishlist.is_selected()) == "True":
```

```
    print("Add to wishes Test Passed")
else:
    print("Add to wishes Test Failed")

driver.quit()
```

8. Delete from wishes Test Chrome:

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.action_chains import ActionChains
import time
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.wait import WebDriverWait

serv_obj = Service("C:/Users/amina/Downloads/БДР/chromedriver-win64/chromedriver.exe")
driver = webdriver.Chrome(service=serv_obj)

driver.get("https://nashformat.ua/")
driver.maximize_window()
time.sleep(1)

enter = driver.find_element(By.ID, "user-menu")
enter.click()
time.sleep(1)
driver.find_element(By.ID, "loginPopup")

email = driver.find_element(By.XPATH, "//*[@id='formLogin']/div[2]/input")
```

```
email.click()
email.send_keys("amina.melnychuk@gmail.com")
time.sleep(1)

password = driver.find_element(By.XPATH, "//*[@id='formLogin']/div[3]/input")
password.click()
time.sleep(1)
password.send_keys("Dk10gh@m")
time.sleep(1)

enter = driver.find_element(By.XPATH, "//*[@id='formLogin']/div[4]/input[2]")
enter.click()
time.sleep(1)

wishlist = driver.find_element(By.XPATH, "//*[@id='wishlist']/a")
wishlist.click()

folders = driver.find_element(By.XPATH, "//*[@id='fn-content']/div[2]/div[1]/div[1]/div/div[1]/select")
folders.click()
time.sleep(2)

shelf = driver.find_element(By.XPATH, "//*[@id='fn-content']/div[2]/div[1]/div[1]/div/div[1]/select/option[2]")
shelf.click()

delete_books = driver.find_element(By.XPATH, "//*[@id='fn-content']/div[2]/div[1]/div[1]/div/div[2]/button")
time.sleep(1)
delete_books.click()
```

```
popup = driver.find_element(By.XPATH, "//*[@id='clear_shelf']/div/div")
time.sleep(2)

yes_button = driver.find_element(By.XPATH, "//*[@id='shelf_clear']/div[2]/input")
yes_button.click()
time.sleep(4)

folders = driver.find_element(By.XPATH, "//*[@id='fn-content']/div[2]/div[1]/div[1]/div/div[1]/select")
folders.click()
time.sleep(2)

shelf = driver.find_element(By.XPATH, "//*[@id='fn-content']/div[2]/div[1]/div[1]/div/div[1]/select/option[2]")
shelf.click()

delete_shelf = driver.find_element(By.XPATH, "//*[@id='fn-content']/div[2]/div[1]/div[2]/div[2]/div[1]/div/div/a[3]")
time.sleep(1)
delete_shelf.click()

popup2 = driver.find_element(By.XPATH, "//*[@id='delete_shelf']/div/div")
time.sleep(2)

yes_button2 = driver.find_element(By.XPATH, "//*[@id='shelf_delete']/div[2]/input")
time.sleep(2)
yes_button2.click()
```

```
if WebDriverWait(driver, 10).until(EC.text_to_be_present_in_element((By.XPATH, "//*[@id='fn-content']/div[2]/div[1]/div[2]/div[1]/div[2]/div/div"), 'Тут збережеться все, що вам сподобалось.')):
    print("Delivery address was successfully added. Test Passed")
else:
    print("Delivery address was not added. Test Failed")

driver.quit()
```

9. Add to wishes Test Edge:

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.action_chains import ActionChains
import time

serv_obj = Service("C:/Users/amina/Downloads/БДР/edge/msedgedriver.exe")
driver = webdriver.Edge(service=serv_obj)

driver.get("https://nashformat.ua/")
driver.maximize_window()
time.sleep(1)

enter = driver.find_element(By.ID, "user-menu")
enter.click()
time.sleep(1)
driver.find_element(By.ID, "loginPopup")

email = driver.find_element(By.XPATH, "//*[@id='formLogin']/div[2]/input")
```

```
email.click()
email.send_keys("amina.melnychuk@gmail.com")
time.sleep(1)

password = driver.find_element(By.XPATH, "//*[@id='formLogin']/div[3]/input")
password.click()
time.sleep(1)
password.send_keys("Dk10gh@m")
time.sleep(1)

enter = driver.find_element(By.XPATH, "//*[@id='formLogin']/div[4]/input[2]")
enter.click()
time.sleep(1)

driver.find_element(By.XPATH, "//*[@id='searchHeaderPlace']")
search = driver.find_element(By.XPATH, "//*[@id='fn-search']/label/input")
search.click()
search.send_keys("Буквар")
time.sleep(2)

find_button = driver.find_element(By.XPATH, "//*[@id='fn-search']/span/button")
find_button.click()
time.sleep(5)

heart = driver.find_element(By.XPATH, "//*[@id='fn-products_content']/div[1]/div/div/div[1]/div/a[2]/i")
driver.execute_script("arguments[0].scrollIntoView();", heart)
time.sleep(2)
driver.execute_script("arguments[0].click();", heart)
time.sleep(2)
```

```
popup = driver.find_element(By.XPATH, "//*[@id='shelvesPopup']/div/div")

shelf = driver.find_element(By.XPATH, "//*[@id='new_shelf_name']")
#driver.execute_script("arguments[0].scrollIntoView();", close_button)
driver.execute_script("arguments[0].click();", shelf)
time.sleep(5)
shelf.send_keys("Дитячі книжки")
time.sleep(5)
#print(close_button.is_selected())
create_button = driver.find_element(By.XPATH,
"//*[@id='shelvesPopup']/div/div/div/div[3]/div[2]/div/form/button")
driver.execute_script("arguments[0].click();", create_button)
time.sleep(5)

wishlist = driver.find_element(By.XPATH, "//*[@id='wishlist']/span")
driver.execute_script("arguments[0].scrollIntoView();", wishlist)
time.sleep(3)
if print(wishlist.is_selected()) == "True":
   print("Add to wishes Test Passed")
else:
   print("Add to wishes Test Failed")

driver.quit()
```

10. Delete from wishes Test Edge:

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.chrome.service import Service
```

```
from selenium.webdriver.common.action_chains import ActionChains
import time
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.wait import WebDriverWait

serv_obj = Service("C:/Users/amina/Downloads/БДР/edge/msedgedriver.exe")
driver = webdriver.Edge(service=serv_obj)

driver.get("https://nashformat.ua/")
driver.maximize_window()
time.sleep(1)

enter = driver.find_element(By.ID, "user-menu")
enter.click()
time.sleep(1)
driver.find_element(By.ID, "loginPopup")

email = driver.find_element(By.XPATH, "//*[@id='formLogin']/div[2]/input")
email.click()
email.send_keys("amina.melnychuk@gmail.com")
time.sleep(1)

password = driver.find_element(By.XPATH, "//*[@id='formLogin']/div[3]/input")
password.click()
time.sleep(1)
password.send_keys("Dk10gh@m")
time.sleep(1)

enter = driver.find_element(By.XPATH, "//*[@id='formLogin']/div[4]/input[2]")
enter.click()
```

```
time.sleep(1)

wishlist = driver.find_element(By.XPATH, "//*[@id='wishlist']/a")
wishlist.click()

folders = driver.find_element(By.XPATH, "//*[@id='fn-content']/div[2]/div[1]/div[1]/div/div[1]/select")
folders.click()
time.sleep(2)

shelf = driver.find_element(By.XPATH, "//*[@id='fn-content']/div[2]/div[1]/div[1]/div/div[1]/select/option[2]")
shelf.click()

delete_books = driver.find_element(By.XPATH, "//*[@id='fn-content']/div[2]/div[1]/div[1]/div/div[2]/button")
time.sleep(1)
delete_books.click()

popup = driver.find_element(By.XPATH, "//*[@id='clear_shelf']/div/div")
time.sleep(2)

yes_button = driver.find_element(By.XPATH, "//*[@id='shelf_clear']/div[2]/input")
yes_button.click()
time.sleep(4)

folders = driver.find_element(By.XPATH, "//*[@id='fn-content']/div[2]/div[1]/div[1]/div/div[1]/select")
folders.click()
time.sleep(2)
```

```
shelf = driver.find_element(By.XPATH, "//*[@id='fn-content']/div[2]/div[1]/div[1]/div/div[1]/select/option[2]")
shelf.click()

delete_shelf = driver.find_element(By.XPATH, "//*[@id='fn-content']/div[2]/div[1]/div[2]/div[2]/div[1]/div/div/a[3]")
time.sleep(1)
delete_shelf.click()

popup2 = driver.find_element(By.XPATH, "//*[@id='delete_shelf']/div/div")
time.sleep(2)

yes_button2 = driver.find_element(By.XPATH, "//*[@id='shelf_delete']/div[2]/input")
time.sleep(2)
yes_button2.click()

if WebDriverWait(driver, 10).until(EC.text_to_be_present_in_element((By.XPATH, "//*[@id='fn-content']/div[2]/div[1]/div[2]/div[1]/div[2]/div/div"), 'Тут збережеться все, що вам сподобалось.')):
    print("Delivery address was successfully added. Test Passed")
else:
    print("Delivery address was not added. Test Failed")

driver.quit()
```

ДОДАТОК В (обов'язковий)

ГРАФІЧНА ЧАСТИНА

«ПРОГРАМНИЙ МОДУЛЬ ТЕСТУВАННЯ WEB-ЗАСТОСУНКІВ»

Виконала: студентка 4 курсу, групи 1КН-20б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Мельничук А. М.
(прізвище та ініціали)
Керівник: к.т.н., доцент каф. КН
Крилик Л. В.
(прізвище та ініціали)

« 07 » червня 2024 р.

Вінниця ВНТУ - 2024 рік

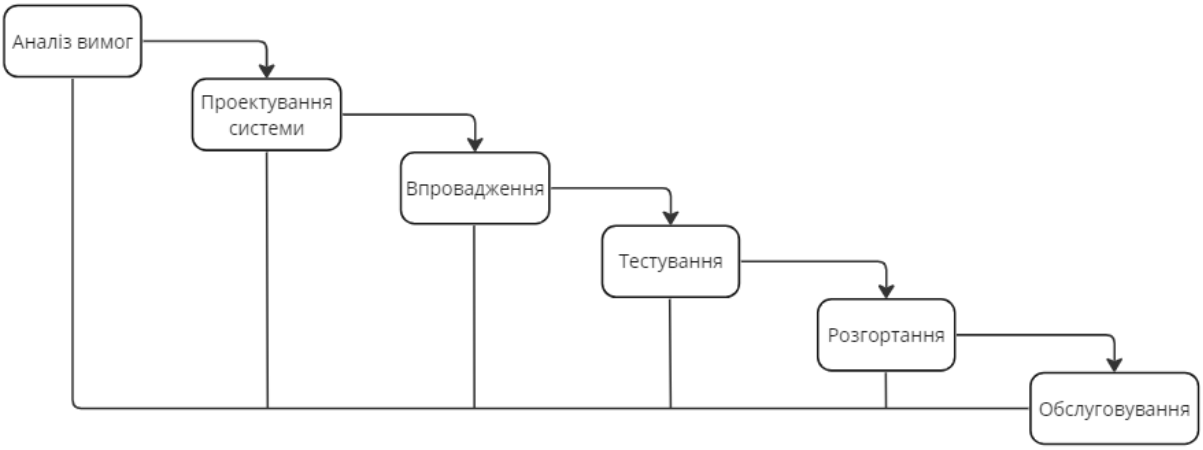


Рисунок В.1 – Фази моделі Водоспад

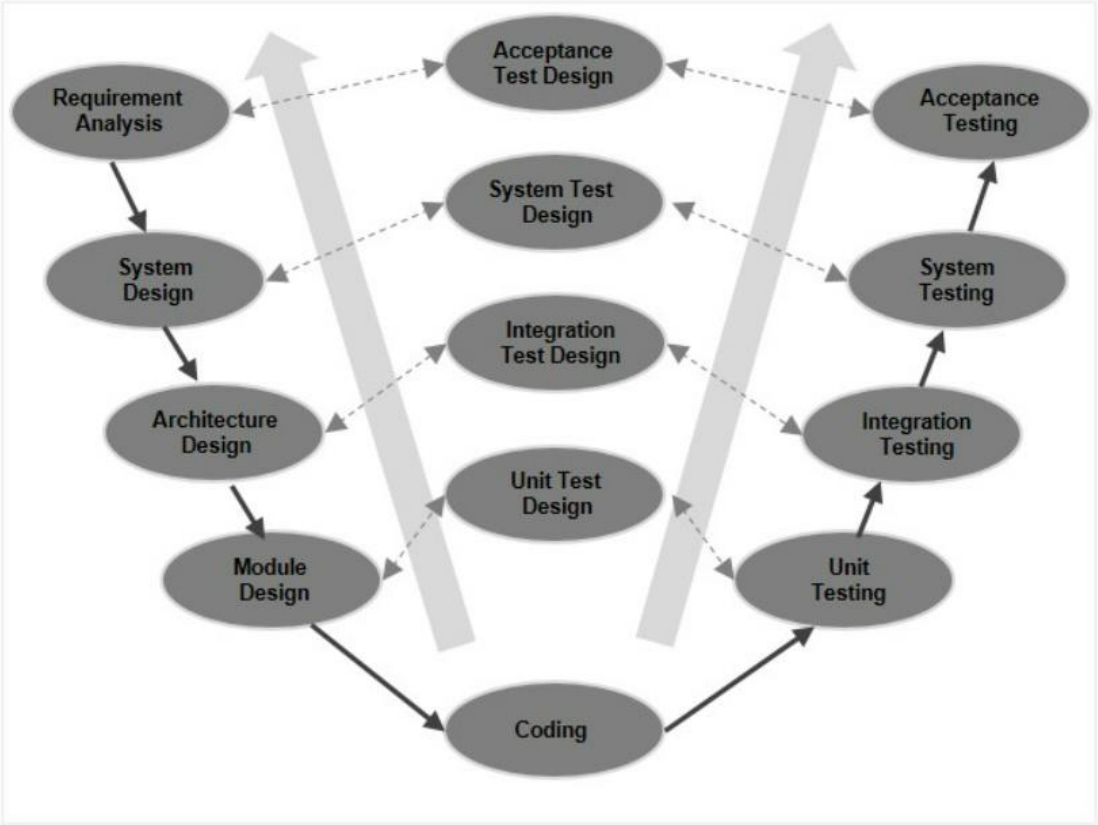


Рисунок В.2 – Фази V-моделі

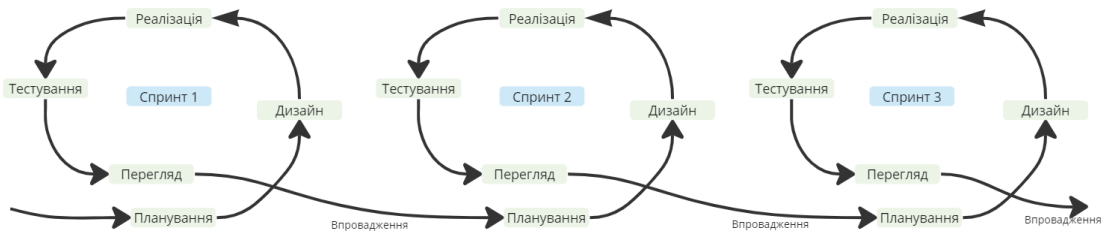


Рисунок В.3 − Фази моделі Agile

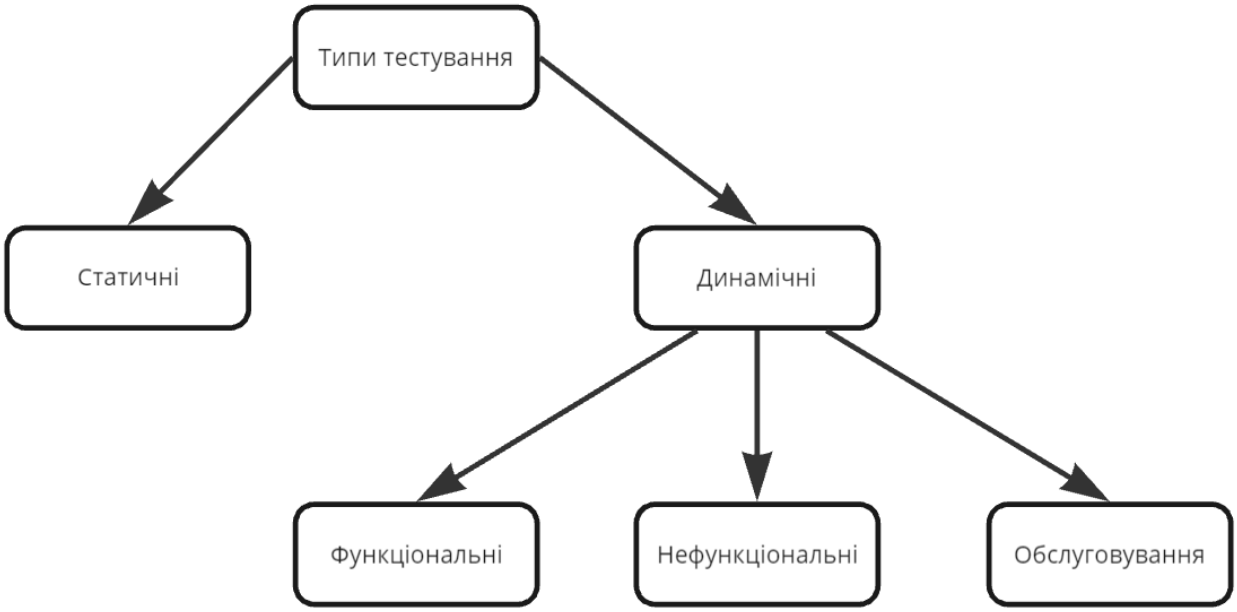


Рисунок В.4 – Рівні та типи тестування

№	Name	Status	Comment
1	Register on the site by entering the number 1 instead of the email	pass	Full name - Білик Настя, Phone number: +38(099)9999999, email: 1, password: ********
2	Register on the site by entering the number 1 instead of full name	fail	Doesn't allow to create account with full name 1 because it was already created. Full name - 1, Phone number: +38(099)9999999, email: rattiroinoda@gmail.com, password: ********
3	Register on the site by entering the word instead of phone number	pass	Full name - Білик Настя, Phone number: number, email: rattiroinoda@gmail.com, password: ********.
4	Log in with a Google Account	pass	
5	Go to the Instagram page	pass	
6	Go to the Youtube channel	pass	
7	Check the title by hovering over the tab	pass	
8	Check the search field	pass	
9	Check whether the top menu bar is fixed	pass	
10	Check whether the vertical scroll works correctly.	pass	
11	Check the work of the back to top button	pass	
12	Check the change of banners	pass	
13	Click on the banner at the top of the page	pass	
14	Watch the video	pass	
15	Exit the personal account	pass	
16	Check the tooltips	pass	
17	Check whether contacts or messenger pages open in a new tab	pass	
18	Go to the book page	pass	
19	Check the carousel of books	pass	
20	Click the button "Всі категорії книг"	pass	
21	Go to "Електронні книжки" category using footer	pass	
22	Expand the full description of the bookstore	pass	
23	Collapse the full bookstore description	pass	
24	Add book to cart	pass	
25	Add a book to favorites	pass	

Рисунок В.5 − Розроблений чек-лист з відповідними статусами

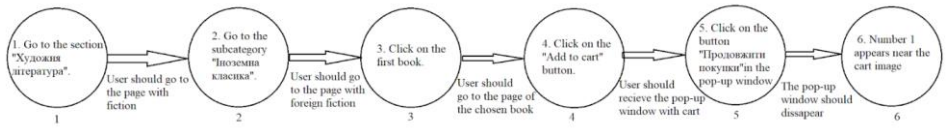
ID 1	
Summary	Add the book to the cart
Precondition	The user is authorized on the site nashformat.ua with the following test data: amina.melnychuk@gmail.com password: ********.
Test design technique	State transition
Steps	1. Go to the section "Художня література". 2. Go to the subcategory "Іноземна класика". 3. Click on the first book. 4. Click the "Add to cart" button. 5. Click the button "Продовжити покупки"in the pop-up window
Expected result	The book is added to cart and number 1 should appear near the cart image in the upper right corner.
Attachment:	

Рисунок В.6 – Тест-кейс про додавання книги до кошика, створений на основі техніки тест дизайну State transition

ID 2						
Summary	Fill in the field Name and Surname in the personal information					
Precondition	1) The user is authorized on the site nashformat.ua with the following test data: amina.melnychuk@gmail.com password: ********.					
	2) The total number of characters in the Name and Last Name field can be from 6 to 12					
Test design technique	Boundary value					
	invalid group		valid group		invalid group	
	-	5	6	12	13	`+
Steps						
	case 1	Enter data in the field name and surname, which have a total of 5 characters				
	Expected result	Must issue an error				
	case 2	Enter data in the field name and surname, which have a total of 6 characters				
	Expected result	Data must be saved				
	case 3	Enter data in the field name and surname, which have a total of 12 characters				
	Expected result	Data must be saved				
	case 4	Enter data in the field name and surname, which have a total of 13 characters				
	Expected result	Must issue an error				

Рисунок В.7 – Тест-кейс про заповнення полів Ім’я та Прізвище в Персональній інформації користувача, створений на основі техніки тест-дизайну Boundary value

ID 4							
Summary	Enter the e-mail for user registration						
Precondition	Go to the site nashformat.ua						
Test design technique	Decision table						
Steps							
	Case	**1**	**2**	**3**	**4**	**5**	**6**
	Should contain @	0	1	0	0	1	1
	Should contain .com	0	1	0	1	1	0
	Written in lower case	0	1	1	0	0	1
	Written in English	0	1	1	1	0	0
	Result	Enter e-mail	Success	Write the correct e-mail, should contain @ and .com	Write the correct e-mail, should be written in lower case and contain @	Write the correct e-mail, should be written in English and in lower case	Write the correct e-mail, should contain .com and be written in English

Рисунок В.8 – Тест-кейс про введення електронної адреси для реєстації акаунта користувача, створений на основі техніки тест дизайну Decision table

ID 5			
Summary	Change the price in the price filter on the Book page		
Precondition	The user is authorized on the site nashformat.ua with the following test data: amina.melnychuk@gmail.com password: ********.		
Test design technique	Equivalence partioning		
	Affordable price from 185 to 468		
	valid group	invalid group	
	185	184	
	468	469	
	359	-185	
	400	50	
	285	100	
Steps	case 1	Enter the minimum price 359	
	Expected result	Books are filtered with a price from 359	
	case 2	Enter the minimum price - 185	
	Expected result	The price filter should not save data and should keep the original values	

Рисунок В.9 – Тест-кейс про зміну ціни в фільтрах на сторінці Книги, створений на основі техніки тест-дизайну Equivalence partioning

ID 6	
Summary	Registration on the website nashformat.ua
Precondition	Go to nashformat.ua
Test design technique	Exploritary testing
Steps	1) Click the "Login" button in the upper right corner.
	2) Change the "Login" tab to the "Registration" tab in the pop-up window
	3) Enter last name and first name: Melnychuk Amina
	4) Enter phone number: +38 (093) 023-11-14
	5) Enter e-mail address: amina.melnychuk@gmail.com
	6) Enter password: ********
	7) Click the "Register" button.
Expected result	The user is registered on the site

Рисунок В.10 – Тест-кейс про реєстрацію на сайті, створений на основі техніки тест-дизайну Exploritary testing

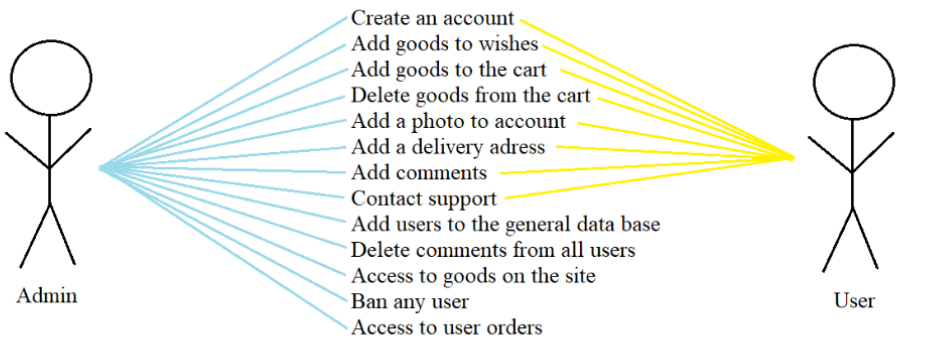
ID 11	
Summary	Explore the differences between user and admin possibilities
Precondition	Go to site nashformat.ua.
Test design technique	Use-case diagram
Attachment:	
Expected result	User shouldn't have the access to the functions of admin

Рисунок В.11 – Тест-кейс про дослідження різниці можливостей користувача та адміністратора, створений на основі техніки тест-дизайну Use-case diagram

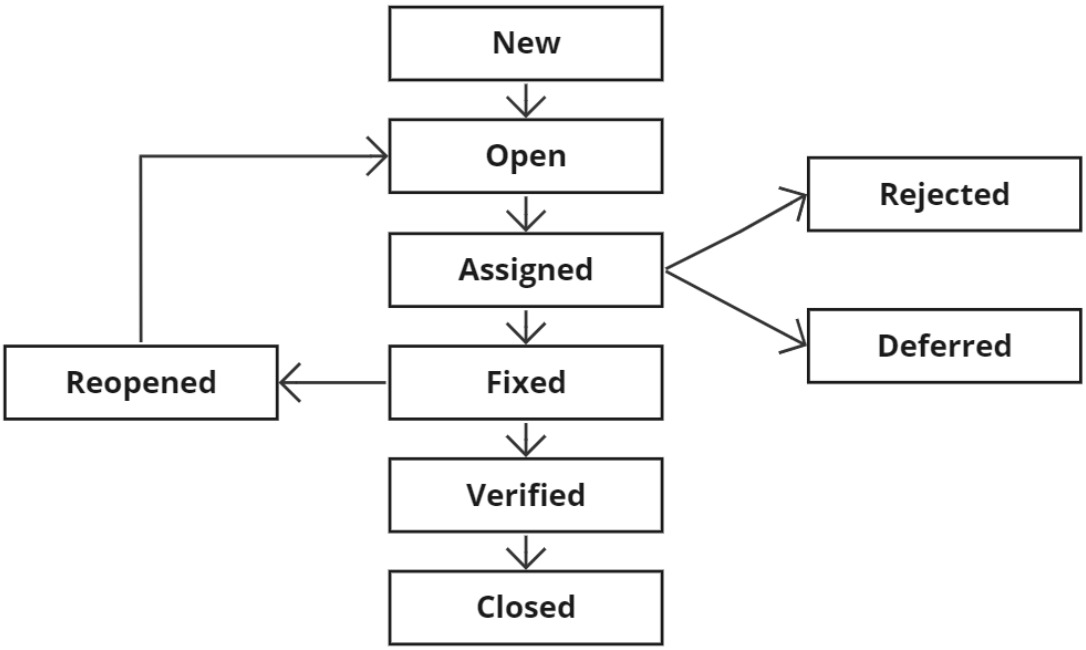


Рисунок В.12 –Життєвий цикл дефекту

ID - 1	
Summary	The user can register on the site with full name 1
Precondition	Go to site nashformat.ua
Steps	1) 1) You click the button "Вхід"
	2) In the pop-up window go to "Реєстрацію"
	3) Enter full name: 1
	4) Enter phone number, email and password: +38(099)9999999, rattiroinoda@gmail.com, ********
	5) Click the button "Зареєструватися"
Expected result	The user will not be able to register an account with name 1 and must receive a message about the invalidity of the entered data in the name field
Actual result	The user receives a message that the user with this name already exists, so it is possible to register with a number instead of a full name
Severity	Major
Priority	Medium
Type	Logical
Assigned to	Dev Eva Silko
Environment:	Windows 11, Google Chrome Version 100.0.4896.88
Attachment:	https://drive.google.com/file/d/1dKQmornIP8g7c568CFVSB46aZmSP5ckN/view?usp=sharing

Рисунок В.13 – Баг-репорт про можливість реєстрації користувача з Іменем 1

ID - 2	
Summary	The user can add a delivery address without specifying a new post office
Precondition	The user is logged in on the site nashformat.ua: tanagulka@gmail.com password: ********.
Steps	1) Hower the mouse over the inscription "Вітаємо, Настя".
	2) In the pop-up window choose "Адреси доставки".
	3) Click the button "Додати адресу".
	4) Click on the image "Нова пошта".
	5) Uder the images choose "Відділення-відділення".
	6) Choose in field "Місто" needed city from the drop down list: Вінниця.
	7) Enter nothing in the field "Відділення"
	8) Click the button "Зберегти".
Expected result	The user should be notified of the lack of data in the field "Відділення"
Actual result	The program automatically selects any branch and adds a delivery address
Severity	Major
Priority	Medium
Type	Functional
Assigned to	Dev Eva Silko
Environment:	Windows 11, Google Chrome Version 100.0.4896.88
Attachment:	https://drive.google.com/file/d/1ThDWhmGByusWtxKmTyeaJRLuCozeHub9/view?usp=sharing

Рисунок В.14 – Баг-репорт про можливість залишити поле відділення пустим при додаванні адреси доставки

ID - 3	
Summary	The name of the subcategory of books overlaps the number of books
Precondition	The user is authorized on the site nashformat.ua with the following test data: amina.melnychuk@gmail.com password: ********.
Steps	1) Click on "Книги" in the top menu
	2) Choose "Історія. Військова справа"
Expected result	The user should see clearly written all the names of the subcategories and the number of books on the left side of the page
Actual result	The inscription Військова справа. Спецслужби overlaps the number og the books in this subcategory
Severity	Minor
Priority	Low
Type	UI
Assigned to	Dev Eva Silko
Environment:	Windows 11, Google Chrome Version 100.0.4896.88
Attachment:	https://drive.google.com/file/d/1ilxTslF-5zwu1gu9D_RNjY0yl0MEANMl/view?usp=sharing

Рисунок В.15 – Баг-репорт про те, що назви підкатегорій книг перекривають кількість книг

ID - 4	
Summary	Wishes counters on books don't work correctly
Precondition	The user is authorized on the site nashformat.ua with the following test data: amina.melnychuk@gmail.com password: ********.
Steps	1) Scroll down the home page
	2) Click on the first bool in section "Нові книги"
	3) Add book to wishes many times by clicking the heart
Expected result	The counter should add and subtract 1 to the number of wishes
Actual result	The counter only subtrects 1 every time you click the heart
Severity	Major
Priority	Medium
Type	Functional
Assigned to	Dev Eva Silko
Environment:	Windows 11, Google Chrome Version 100.0.4896.88
Attachment:	https://drive.google.com/file/d/11xUpUnbLP1dhXkBRXvsMiFjRgD0qBEoL/view?usp=sharing

Рисунок В.16 – Баг-репорт про те, що кнопки додати до побажань працюють некоректно

ID - 5	
Summary	Drop down list works incorect
Precondition	The user is authorized on the site nashformat.ua with the following test data: amina.melnychuk@gmail.com password: ********.
Steps	1) Go to section "Книги"
	2) Choose category "Електронні книги"
	3) On the left side in the filter menu close and open drop down lists
Expected result	Drop down list should open smoothly
Actual result	Drop down list opens sharply
Severity	Major
Priority	Medium
Type	Functional
Assigned to	Dev Eva Silko
Environment:	Windows 11, Google Chrome Version 100.0.4896.88
Attachment:	https://drive.google.com/file/d/1UiSBNmFvi785CaYcv7-5AFSJT1dMaJB6/view?usp=sharing

Рисунок В.17 – Баг-репорт про те, що випадаючий список працює некоректно

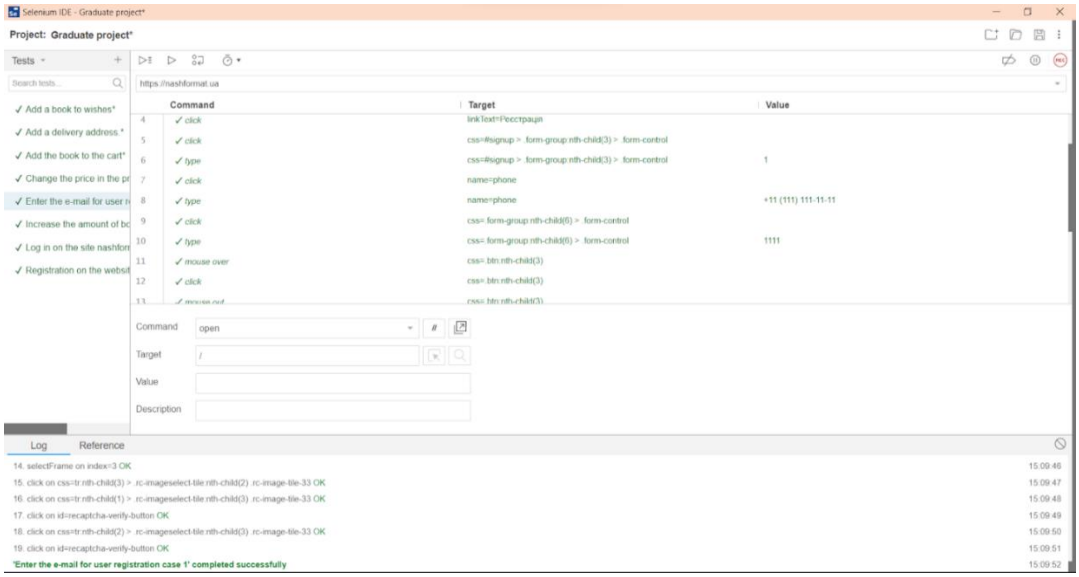


Рисунок В.18 − Створені автоматизовані тести в Selenium IDE

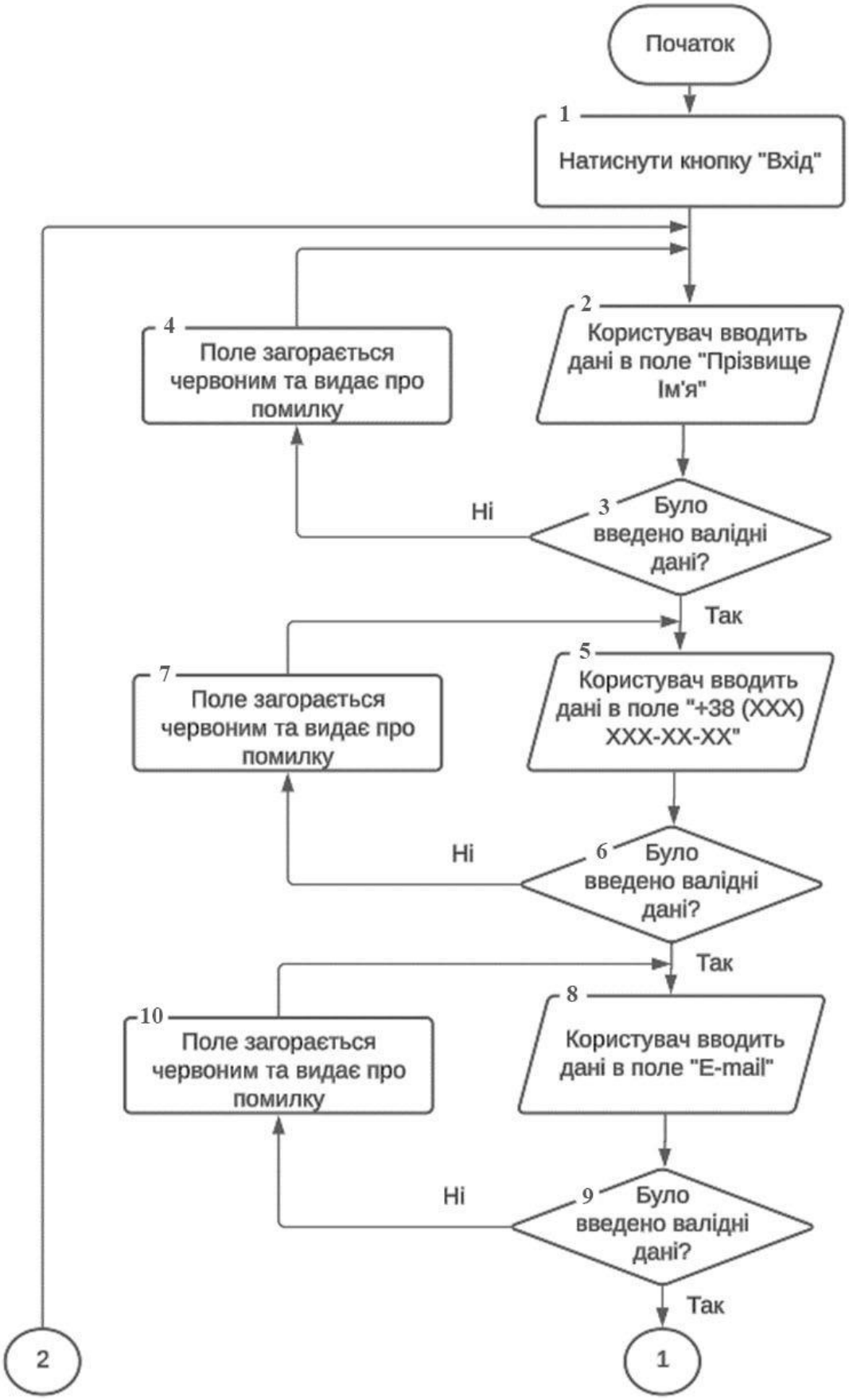


Рисунок В.19 – Схема алгоритму перевірки введення початкових даних

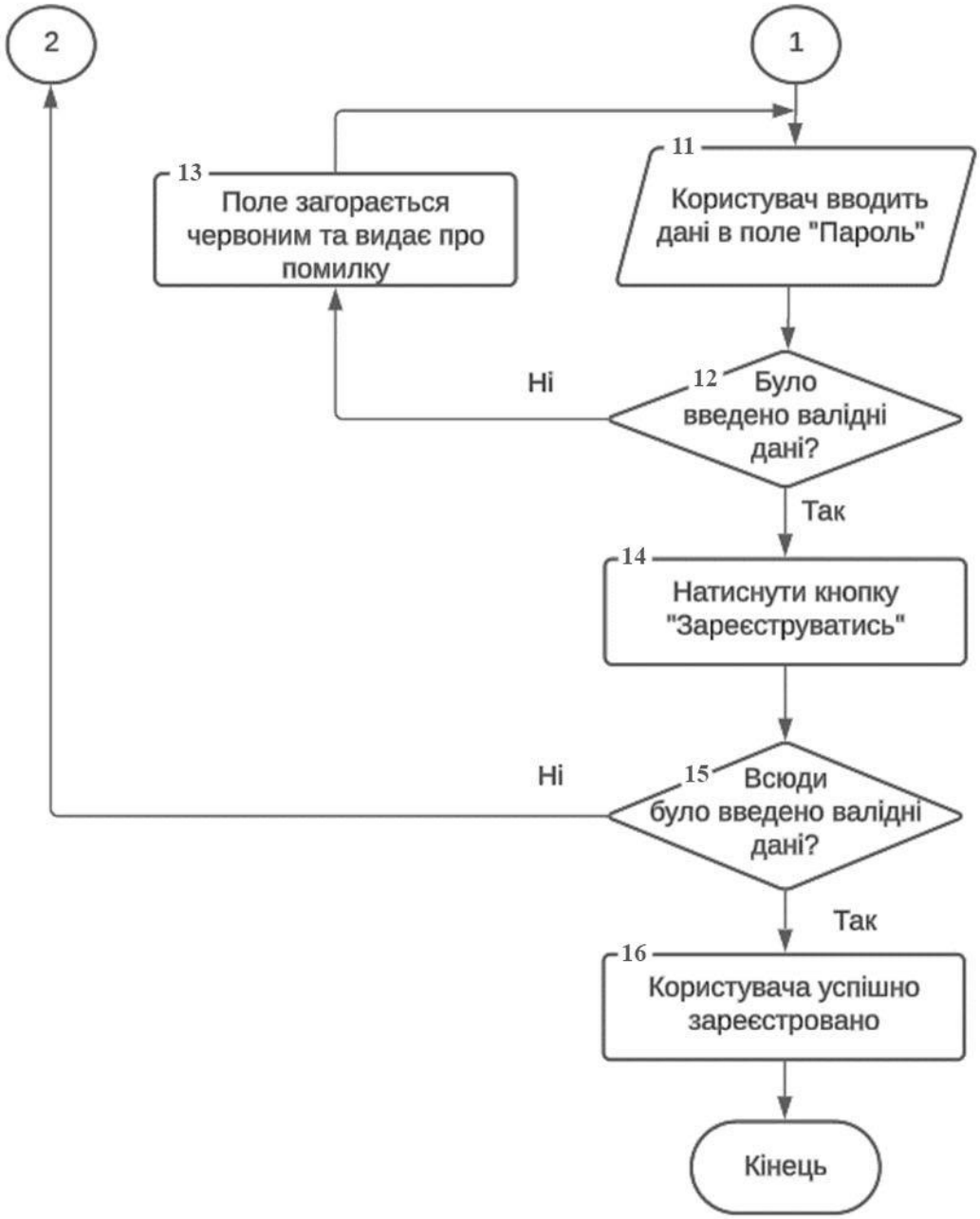


Рисунок В.19, аркуш 2

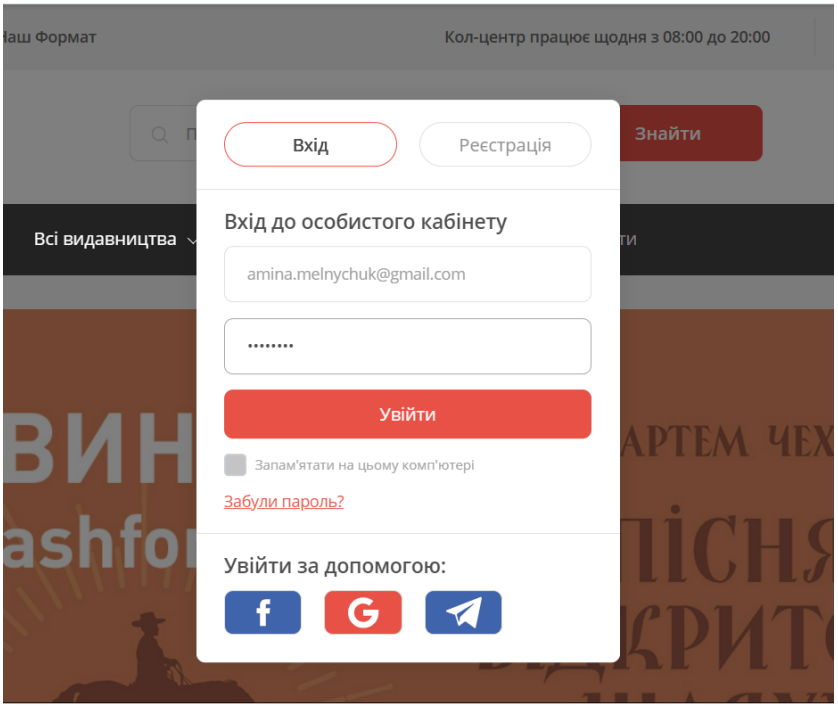


Рисунок В.20 – Вікно введення даних з валідними значеннями

```
driver.find_element(By.ID, value: "loginPopup")

email = driver.find_element(By.XPATH, value: "//*[@id='formLogin']/div[2]/input")
email.click()
email.send_keys("amina.melnychuk@gmail.com")
time.sleep(5)

password = driver.find_element(By.XPATH, value: "//*[@id='formLogin']/div[3]/input")
password.click()
time.sleep(2)
password.send_keys("Dk10gh@m")
time.sleep(5)

enter = driver.find_element(By.XPATH, value: "//*[@id='formLogin']/div[4]/input[2]")
```

```
Login Test Chrome
C:\Users\amina\AppData\Local\Programs\Python\Python312\python.exe "C:\Users\amina\Downloads\БДР\Login Test Chrome.py"
Login Test Passed

Process finished with exit code 0
```

Рисунок В.21 – Вікно з отриманим результатом "Login Test Passed"

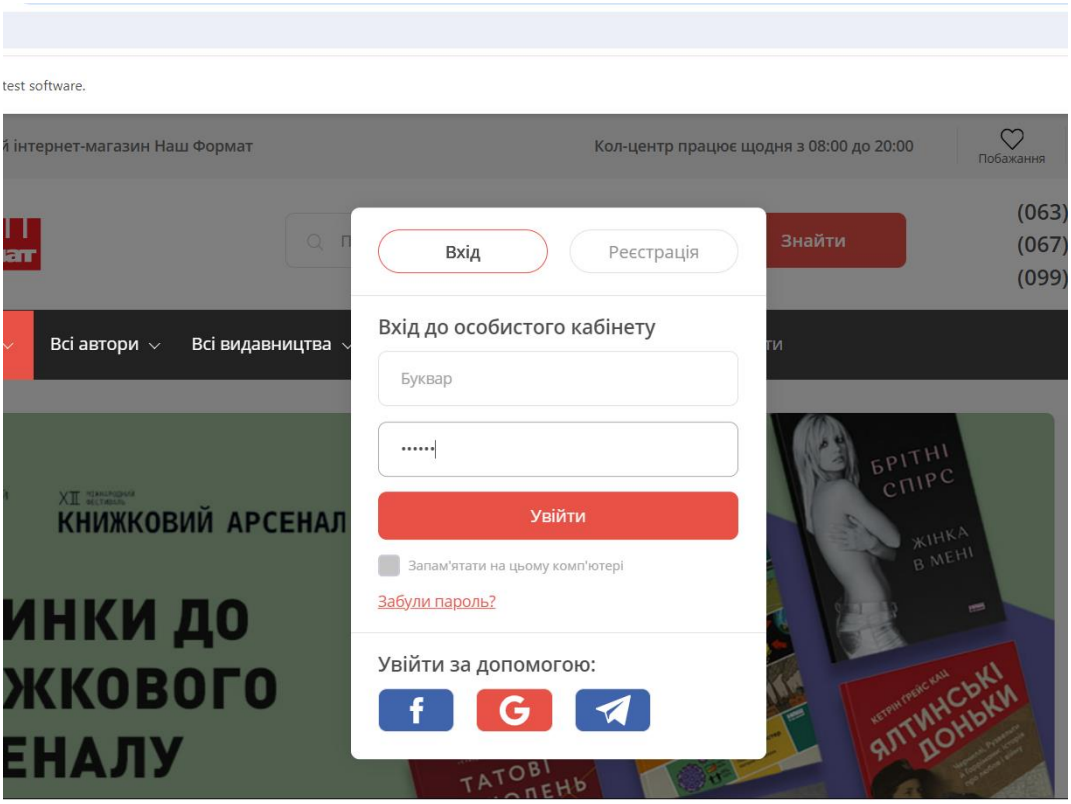


Рисунок В.22 – Вікно введення даних з неправильними значеннями

```
email = driver.find_element(By.XPATH, value: "//*[@id='formLogin']/div[2]/input")
email.click()
email.send_keys("Буквар")
time.sleep(5)

password = driver.find_element(By.XPATH, value: "//*[@id='formLogin']/div[3]/input")
password.click()
time.sleep(2)
password.send_keys("Буквар")
time.sleep(5)

enter = driver.find_element(By.XPATH, value: "//*[@id='formLogin']/div[4]/input[2]")
enter.click()
```

```
Login Test Chrome
C:\Users\amina\AppData\Local\Programs\Python\Python312\python.exe "C:\Users\amina\Downloads\БДР\Login Test Chrome.py"
Login Test Failed

Process finished with exit code 0
```

Рисунок В.23 – Вікно з отриманим результатом "Login Test Failed"

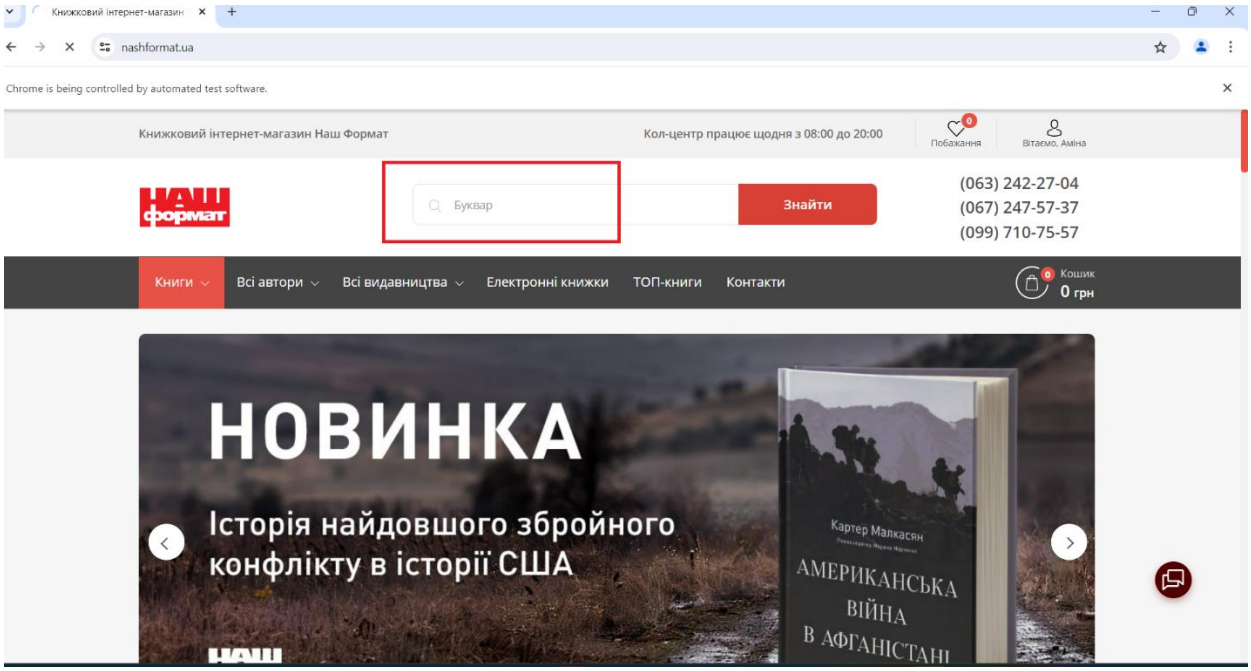


Рисунок В.24 – Вікно введення даних в поле Пошук

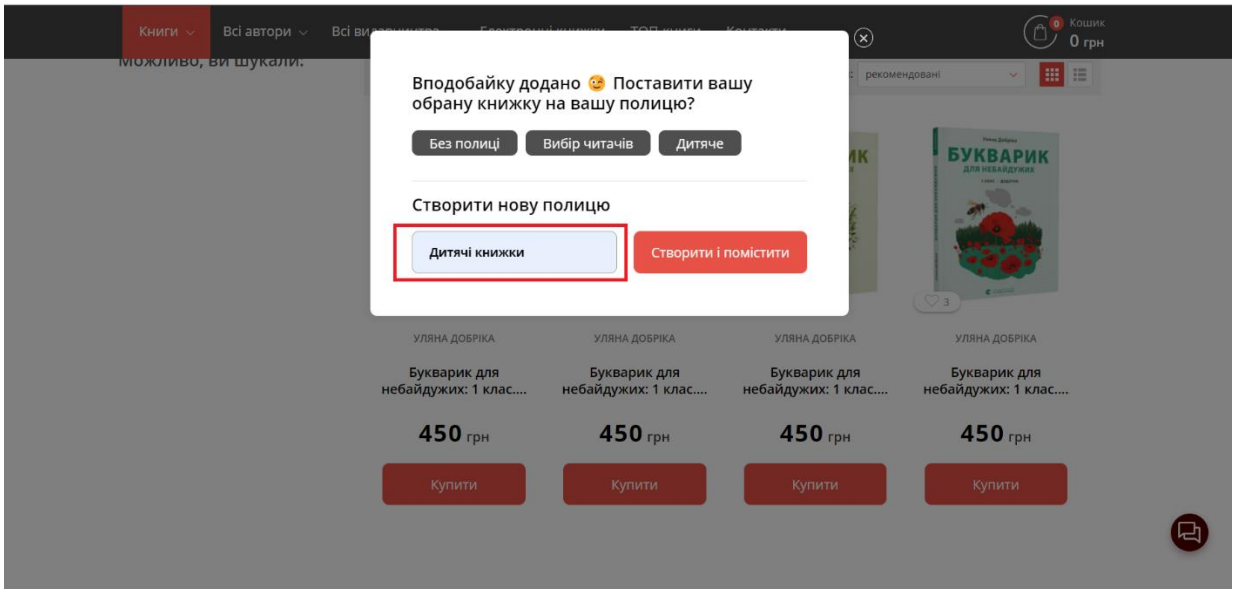


Рисунок В.25 – Вікно створення нової полиці для обраної книги

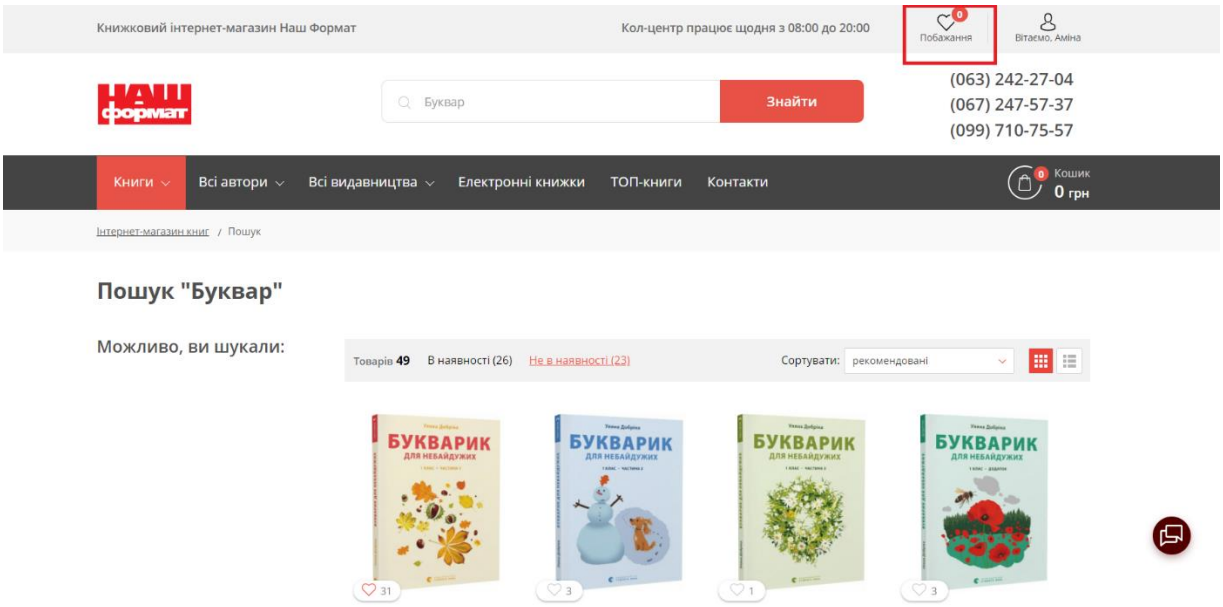


Рисунок В.26 – Показник кількості вподобаних книг

Рисунок В.27 – Вікно з отриманим результатом "Add to wishes Test Failed"

ДОДАТОК Г (довідниковий)
Інструкція користувача

Для початку роботи потрібно налаштувати відповідне середовище. Перш за все потрібно завантажити файли драйверів, в нашому випадку це будуть файли для Google Chrome та Microsoft Edge. Усі файли драйверів потрібно розархівувати для подальшої роботи з ними. Для реалізації автоматизованих тестів мовою Python за допомогою Selenium WebDriver потрібно завантажити програму PyCharm та завантажити сам Python. Під час налаштувань PyCharm обираємо потрібний файл з Python в ролі інтерпретатора та завантажуємо пакет під назвою Selenium.

Потім відкриваємо розроблений файл з тестом, наприклад, Add delivery address Test Chrome. На рисунку Г.1 подано відкритий файл з тестом в середовищі PyCharm.

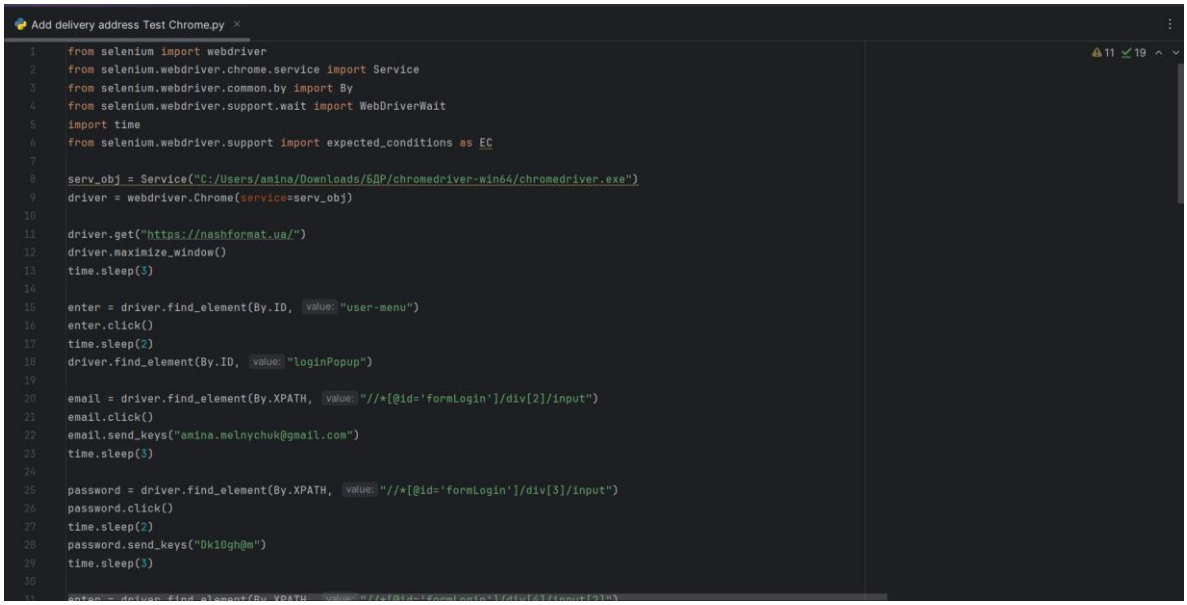


Рисунок Г.1 – Файл з потрібним автоматизованим тестом
у середовищі PyCharm

В файлі шукаємо елемент під назвою search_city (65 рядок коду) і в полі search_city.send_keys (69 рядок коду) пишемо потрібне місто доставки, наприклад, Вінниця. На рисунку Г.2 представлено потрібні фрагменти коду з новими даними.

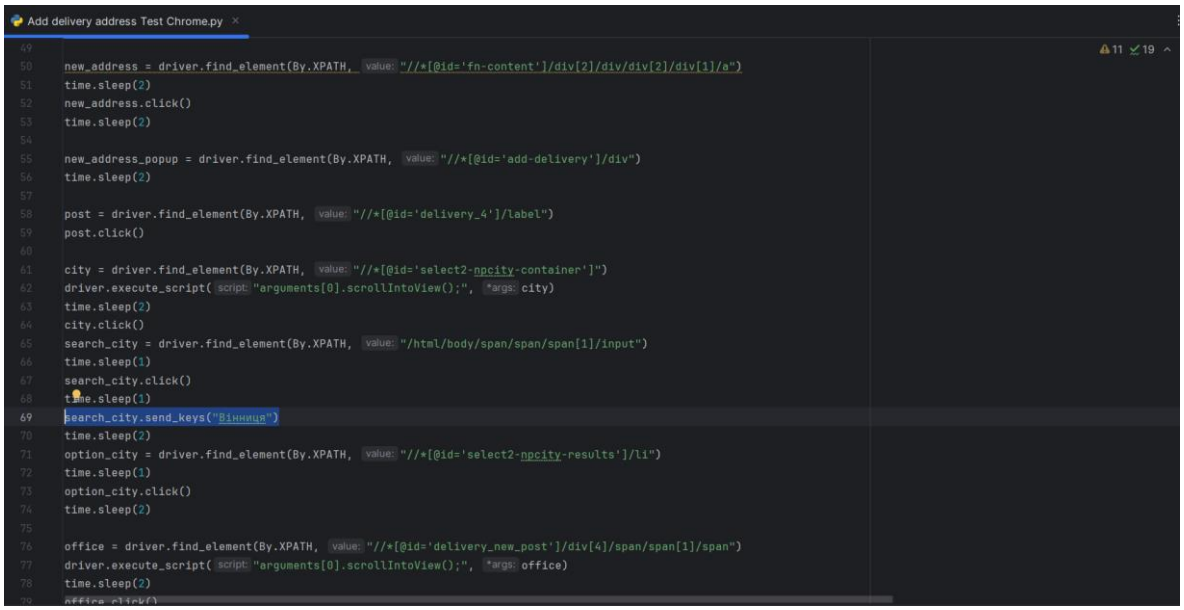


Рисунок Г.2 – Фрагмент коду для вибору потрібного міста доставки

Потім обираємо потрібне відділення: шукаємо елемент під назвою search_office (82 рядок коду) та в полі search_office.send_keys (86 рядок коду) пишемо потрібне відділення, наприклад, 12. На рисунку Г.3 представлено потрібні фрагменти коду з новими даними.

Рисунок Г.3 – Фрагмент коду для вибору потрібного відділення

Тепер натискаючи на кнопку Run в верхньому правому кутку запускаємо автоматизований тест з потрібними даними.

Спершу буде відкрито вікно головної сторінки сайту, а потім спливаюче вікно входу в акаунт користувача, яке представлене на рисунку Г.4.

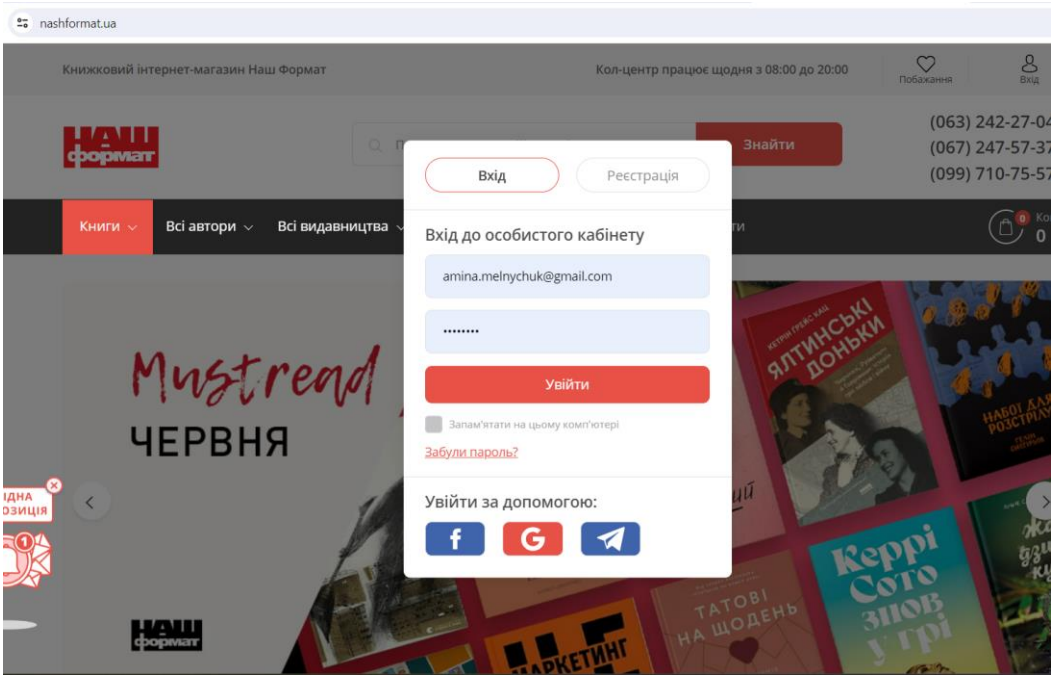


Рисунок Г.4 – Спливаюче вікно входу в акаунт користувача

Після входу в акаунт автотест переадресовує на сторінку «Особистий кабінет» (рис. Г.5).

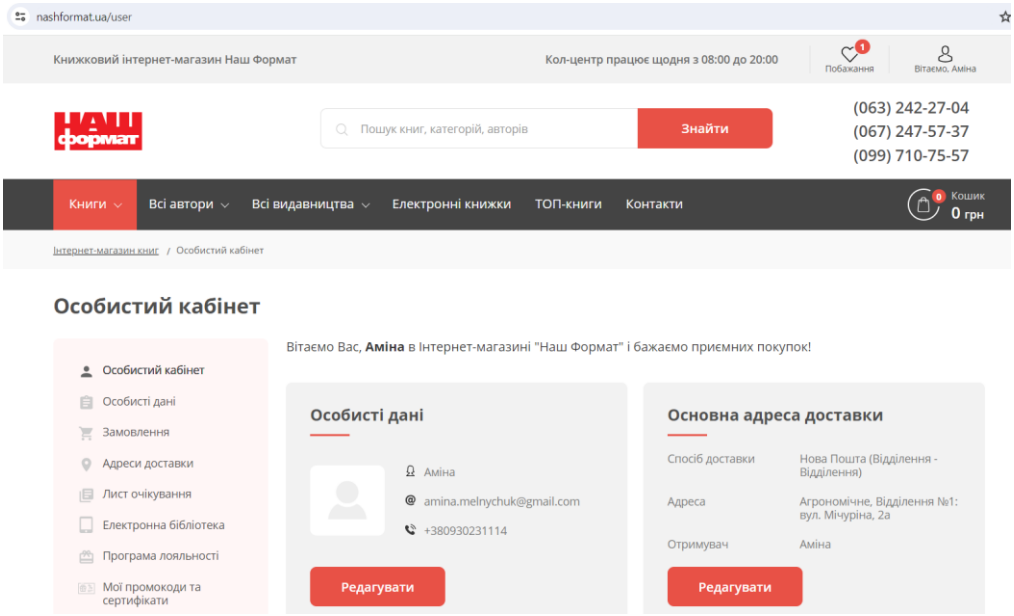


Рисунок Г.5 – Сторінка «Особистий кабінет»

Потім тест переносить нас на сторінку «Адреси доставк» (рис. Г.6).

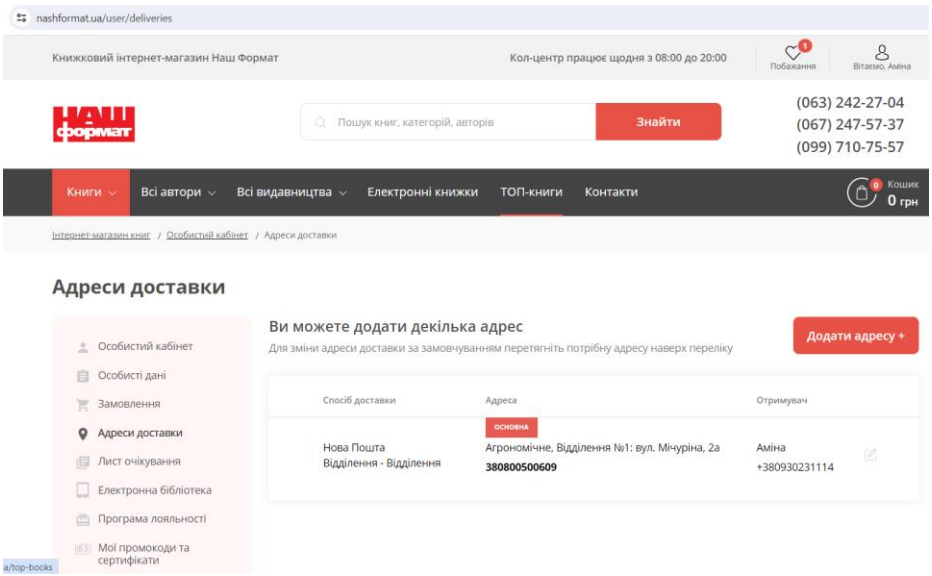


Рисунок Г.6 − Сторінка «Адреси доставки»

Автоматизований тест натискає на кнопку «Додати адресу» та вводить потрібні дані. На рисунку Г.7 представлене спливаюче вікно з введеними даними.

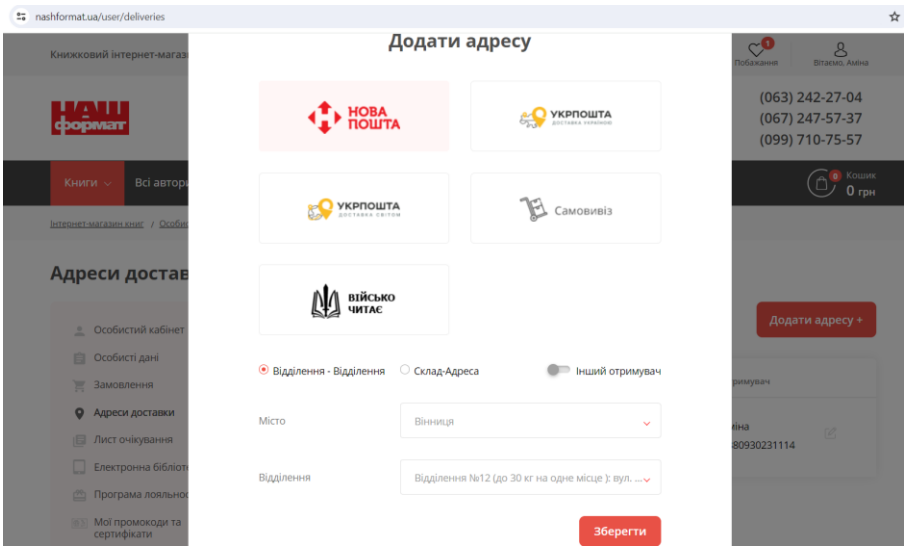


Рисунок Г.7 – Спливаюче вікно додавання нової адреси з введеними даними

Після натискання кнопки «Зберегти», автотест перевіряє відображення нової адреси на сторінці «Адреси доставки». На рисунку Г.8 представлено успішне додавання нової адреси доставки.

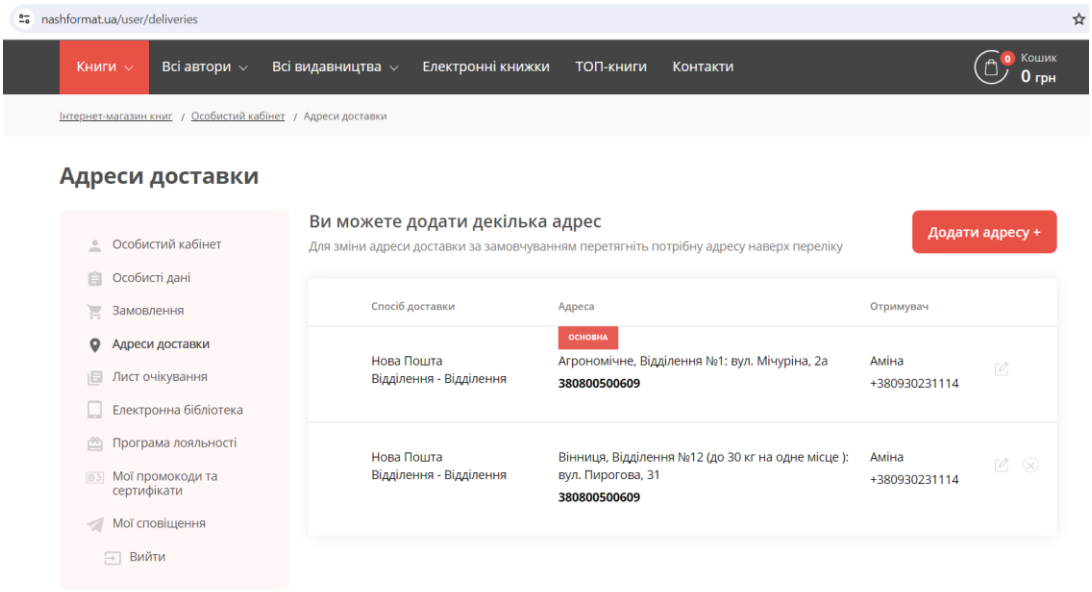


Рисунок Г.8 – Успішно додана нова адреса доставки

Після завершення роботи тесту отримуємо відповідний результат в залежності від того чи успішно додалась адреса, чи ні. На рисунку Г.9 подано результат «Delivery address was successfully added. Test Passed», це свідчить про те, що нова адреса доставки була успішно додана.

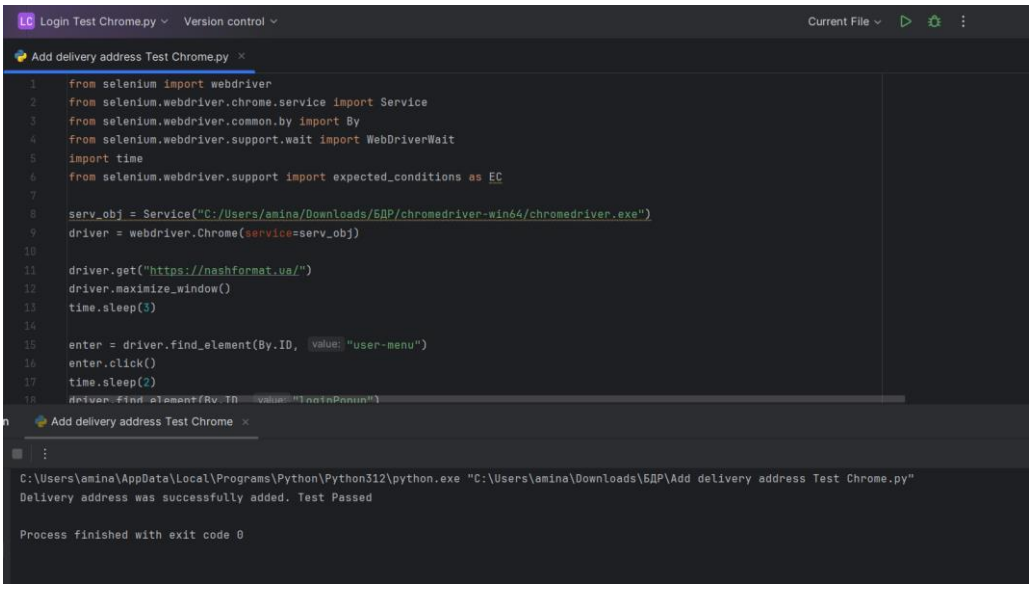


Рисунок Г.9 – Результат роботи програми

Для реалізації всіх інших рохроблених тестів виконуємо такі самі дії.