МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«АВТОМАТИЗАЦІЯ ТЕСТУВАННЯ ІНТЕРНЕТ-МАГАЗИНУ З ВИКОРИСТАННЯМ CYPRESS»**

Виконав: студент 2 курсу, групи 8.1211-1іпз
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
М.О. Рейда
(ініціали та прізвище)

Керівник професор кафедри програмної інженерії, професор, д.т.н. Чопоров С.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

« ____ » _____ 2022 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Рейді Максиму Олександровичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Автоматизація тестування інтернет-магазину з використанням Cypress

керівник роботи Чопоров Сергій Вікторович, д.т.н., професор
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 04 » травня 2022 року № 500-с

2. Строк подання студентом роботи

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік питань до розробки.
3. Основна література.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі, аналіз предметної області.
2. Аналіз вимог до інформаційної системи.
3. Реалізація інформаційної системи.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

Дата видачі завдання 04.05.2022

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	10.05.2022	
2.	Збір вихідних даних.	14.05.2022	
3.	Обробка методичних та теоретичних	09.06.2022	
	джерел.		
4.	Розробка першого розділу.	11.07.2022	
5.	Розробка другого розділу.	05.08.2022	
6.	Розробка третього розділу.	16.09.2022	
7.	Оформлення та нормоконтроль	25.11.2022	
	кваліфікаційної роботи.		
8.	Захист кваліфікаційної роботи.	16.12.2022	

Студент _____ М.О. Рейда
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.В. Чопоров
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Автоматизіція тестування інтернет-магазину з використанням Cypress»: 59 с., 17 рис., 4 табл., 9 джерел.

АВТОМАТИЗАЦІЯ, АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ, ІНТЕРНЕТ-МАГАЗИН, CYPRESS, JENKINS, MOCHA.

Об’єкт дослідження – автоматизація тестування інтернет-магазину.

Предмет дослідження – фреймворк автоматизованного тестування Cypress.

Мета роботи: розробити та реалізувати систему автоматичного тестування інтерент-магазину.

Методи дослідження: аналітичний, моделювання, проектування, тестування продукту.

У кваліфікаційній роботі розглянуті методи та засоби тестування. Розглянуті характеристики Cypress. Проведено порівняльній аналіз з суміжними технологіями. Розроблено тест кейси. Розроблені діаграми. Проведено тестування.

SUMMARY

Master’s Qualifying Paper «Online Store Tesing Automation using Cypress»: 59 pages, 17 figures, 4 tables, 9 references.

AUTOMATION, AUTOMATION TESTING, ONLINE STORE, CYPRESS, JENKINS, MOCHA.

The object of the study is the automation testing of the online store.

The subject of research – framework of automation testing Cypress.

The aim of the study is to develop the automation testing system for the online store.

The methods of research are analytical, modeling, design, product testing.

The methods and means of testing are considered in the qualification work. Cypress features reviewed. A comparative analysis with related technologies. Test cases have been developed. Developed diagrams. Testing was performed.

ЗМІСТ

ВСТУП

У сучасному світи буде доволі складно знайти людину, яка б не робила покупки в Інтернеті. Електронна комерція – це бізнес який процвітає завдяки своїм клієнтам. Протягом 2021 року більше двох мільярдів людей купували товари онлайн. У 2022 році продажі в інтернет магазинах перевищили 5.7 трильйона доларів по всьому світу. Це зростання продовжуватиметься невпинно в осяжному майбутньому, і тому це вимагає впровадження сучасних інструментів, таких як автоматизація тестування.

Забезпечення якості – невід'ємна частина циклу розробки програмного забезпечення. Після кожної зміни програмного продукту є необхідність виконувати тестування. Вартість знайденої під час розробки помилки не є дуже великою, тому що вона може бути доволі швидко виправлена і вона не впливає на роботу користувача. Але як що помилка була виявлена клієнтом, вартість її виправлення зростає у рази, а інколи вона може коштувати життя, якщо ми кажемо про медицину або військову сферу. Саме тому, більшість IT-компаній витрачають великі бюджети на тестування свого програмного продукту.

Розрізняють два основних виду тестування – ручне та автоматизоване. Велика кількість компаній бажає автоматизувати процес тестування у себе, але повністю відійти від ручного тестування не вийде. Трапляються випадки, коли щось потрібно перевірити один раз і витрачати час та гроші на автоматизацію просто не має сенсу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Історія тестування

Тестування програмного забезпечення з’явилося разом із розробкою програмного забезпечення, яка почалася одразу після Другої світової війни. Комп’ютерний вчений Том Кілберн вважається автором першої частини програмного забезпечення, яка дебютувала 21 червня 1948 року в Манчестерському університеті в Англії. Він виконував математичні розрахунки за допомогою інструкцій машинного коду.

Налагодження було основним методом тестування в той час і залишалося ним протягом наступних двох десятиліть. До 1980-х років команди розробників не лише виявляли та виправляли помилки програмного забезпечення, а й тестували програми в реальних умовах. Це заклало основу для ширшого погляду на тестування, яке охоплювало процес забезпечення якості, який був частиною життєвого циклу розробки програмного забезпечення.

У 1990-х роках відбувся перехід від тестування до більш комплексного процесу під назвою забезпечення якості, який охоплює весь цикл розробки програмного забезпечення та впливає на процеси планування, проектування, створення та виконання тестових сценаріїв, підтримки існуючих тестових сценаріїв і тестування середовищ. Тестування вийшло на якісно новий рівень, що призвело до подальшого розвитку методологій, появи потужних інструментів для управління процесом тестування та засобів автоматизації тестування.

1.2 Тестування програмного забезпечення

Розробка програмного забезпечення є тривалим і доволі складним процесом, він вимагає використання різних засобів та типів тестування програмного забезпечення. Методи тестування змінюються з часом. Оскільки область тестування програмного забезпечення зростає, вона стає все більш складною. При автоматизованому тестуванні збільшується швидкість розробки, економиться час, людські ресурси та гроші.

Тестування використовується в галузі розробки, як забезпечення якості, різних частин програмного проекту. Цей процес виконується не лише на завершальних стадіях розвитку, а й на всіх етапах різними способами. Наприклад, існують стратегії розробки, засновані на використанні тестів, які визначають вимоги, а потім змінюють програму, поки вона не пройде тести. В інших випадках використовується тестування програмного забезпечення, яке постійно оновлюється та випускається в нових версіях, щоб перевірити, чи все ще працюють ті частини програмного забезпечення, які раніше працювали.

1.2.1 Рівні тестування

Багато команд вважають, що програмне забезпечення слід тестувати лише після його завершення. Ця думка випливає з методології тестування програмного забезпечення Waterfall – застарілого процесу, який створює більше проблем, ніж вирішує. У більшості випадків, коли рівні тестування програмного забезпечення ігноруються, програмне забезпечення має більше помилок, виправлення яких коштує дорожче, ніж якщо б вони були знайдені раніше. Це означає більше роботи для розробників і більше витрат для компанії, оскільки вона повинна покривати додаткові години, витрачені на виправлення багів. Лише в США

вартість програмних помилок становить приблизно 2,8 трильйона доларів на рік. Але це ще не вся проблема. Уявіть собі, що може статися у разі збоїв програмного забезпечення, які вплинуть не лише на фінанси споживачів, але й на їх життя [1].

Є 4 основні рівні тестування програмного забезпечення, які необхідно виконати перед запуском програмного забезпечення. Ці тести можуть проводитися власними силами розробників в організації або передаватися тестувальникам програмного забезпечення [1].

Рівні тестування включають:

- модульне тестування;
- інтеграційне тестування;
- системне тестування;
- приймальне тестування.

На рисунку 1.1 зображено 4 рівні тестування програмного забезпечення: від тестування окремих блоків, тестування інтегрованих блоків, тестування всієї системи до кінцевого рівня, на якому перевіряється прийнятність усієї системи [1].

Модульне тестування – це перший рівень тестування програмного забезпечення. Тут окремі рядки коду, різні функції та бажані процедури виділяються та тестуються. Ці рядки коду, функції та процедури називаються програмними одиницями, оскільки вони об’єднані, щоб скласти програмне забезпечення. Їх також можна назвати компонентами програмного забезпечення.

Наприклад, рядок коду в «програмному забезпеченні калькулятора» може мати функцію реалізації «додавання», тоді як інший рядок коду реалізує «множення» тощо. Ці функції додавання та множення та інші функції програмного забезпечення калькулятора повинні бути індивідуально перевірені, щоб переконатися, що калькулятор працює бездоганно.

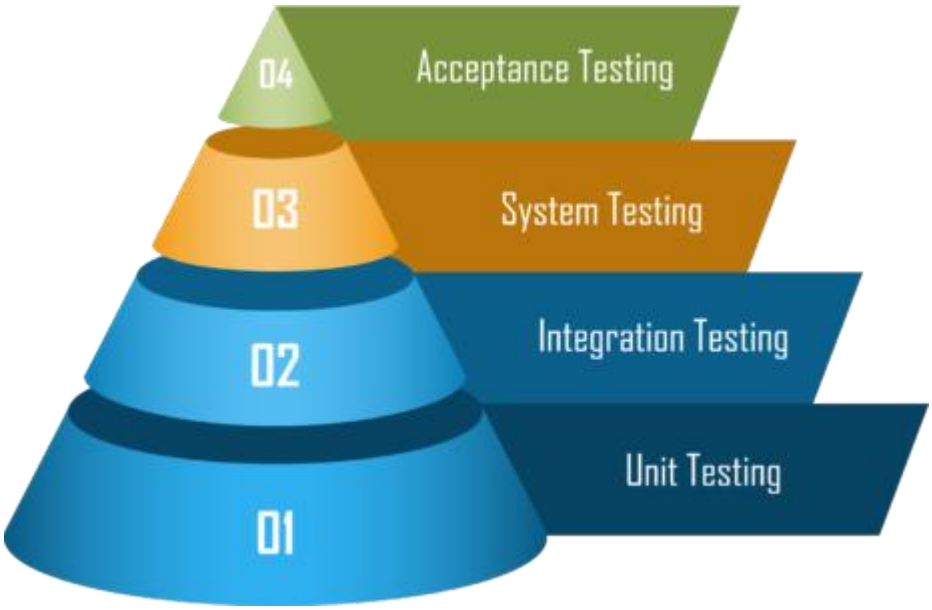

Рисунок 1.1 – Схема рівнів тестування

Користувач не захоче запускати калькулятор, який реалізує ділення, коли користувач натискає на знак додавання. Модульне тестування – це процес, який здебільшого включає тестування внутрішньої роботи програмного забезпечення. Зробити це легко та швидко, оскільки йдеться про програмне забезпечення окремо, а не загалом.

Інтеграційне тестування – цей рівень тестування передбачає об’єднання всіх компонентів, які складають програмне забезпечення, і тестування всього в цілому, а не окремо, як це робилося під час модульного тестування. Крім того, з цього рівня тести можна розділити на функціональні та нефункціональні типи.

Інтегровані коди могли бути написані різними розробниками, і їх потрібно протестувати, щоб переконатися в їх правильності. За допомогою тестування ми можемо виявити та виправити дефекти взаємодії, дефекти одночасної роботи, дефекти паралельної роботи тощо.

Простіше кажучи, це допомагає нам визначити, наскільки добре пристрої працюють разом і стан інтерфейсів між кожним із них.

Наприклад, ми можемо перевірити, чи виконуються такі умови з

контрольного списку:

– чи здійснюється зв'язок між системами правильно;

– чи можуть зв’язані документи безперебійно працювати на всіх платформах;

– чи працюють специфікації безпеки під час зв’язку між системами;

– чи може програмне забезпечення витримувати збої мережі між веб-серверами та серверами програм.

Стан інтерфейсу може бути незадовільним, тому програмному забезпеченню може знадобитися багато часу для перемикання між функціями. Хороший інтерфейс перемикається між функціями відразу після того, як користувач її запускає. Іноді рядок коду може працювати належним чином, якщо його перевірити окремо. Однак у поєднанні з іншими рядками коду для досягнення потрібної функції може виникнути помилка, яка вказує на те, що інтеграція одного або кількох рядків коду в решту коду була невдалою.

Модульне тестування може довести, що все працює, як очікувалося, тоді як інтеграційне тестування може довести протилежне. Ось чому всі рекомендовані рівні тестування програмного забезпечення мають виконуються суворо, щоб уникнути потенційних пасток.

На рисунку 1.2 відображено 4 основні підходи, які використовуються для проведення інтеграційного тестування для наших клієнтів. Вони включають:

1) тестування інтеграції «зверху вниз» (Top-Down);
2) інтеграційне тестування «знизу вгору» (Bottom-Up);
3) інтеграційне тестування великого вибуху (Big Bang);
4) змішане (Mixed) інтеграційне тестування.

Тестування інтеграції «зверху вниз» (*Top-Down*). Під час застосування підходу «зверху вниз» емпіричне правило полягає в тому, щоб перевірити вищі одиниці перед перевіркою нижчих рівнів. Іншими словами, виконавець повинен тестувати зверху вниз.

Інтеграційне тестування «знизу вгору» (*Bottom-Up*). З іншого боку, при інтеграції «знизу вгору» виконавець тестує в протилежному напрямку – знизу вгору. Зрештою ви пробуєте нижчі одиниці та переходите до вищих компонентів.

Інтеграційне тестування великого вибуху *(Big Bang)*. Інтеграційне тестування великого вибуху означає, коли всі елементи тестуються одночасно в одній фазі. Модель великого вибуху також застосовна під час перевірки невеликих програм, де можна обійти моделі «зверху вниз» і «знизу вгору». Усі модулі програмуються, комбінуються та тестуються разом.

Змішане (*Mixed*) інтеграційне тестування: доступні модулі тестуються незалежно від того, чи належать вони до високого чи низького рядків коду. Це означає, що виконавець не надає пріоритету жодному компоненту. Модулі, що не відповідають, імітуються, щоб визначити, наскільки добре працюватиме доступний код, коли кодування завершено та об’єднано.

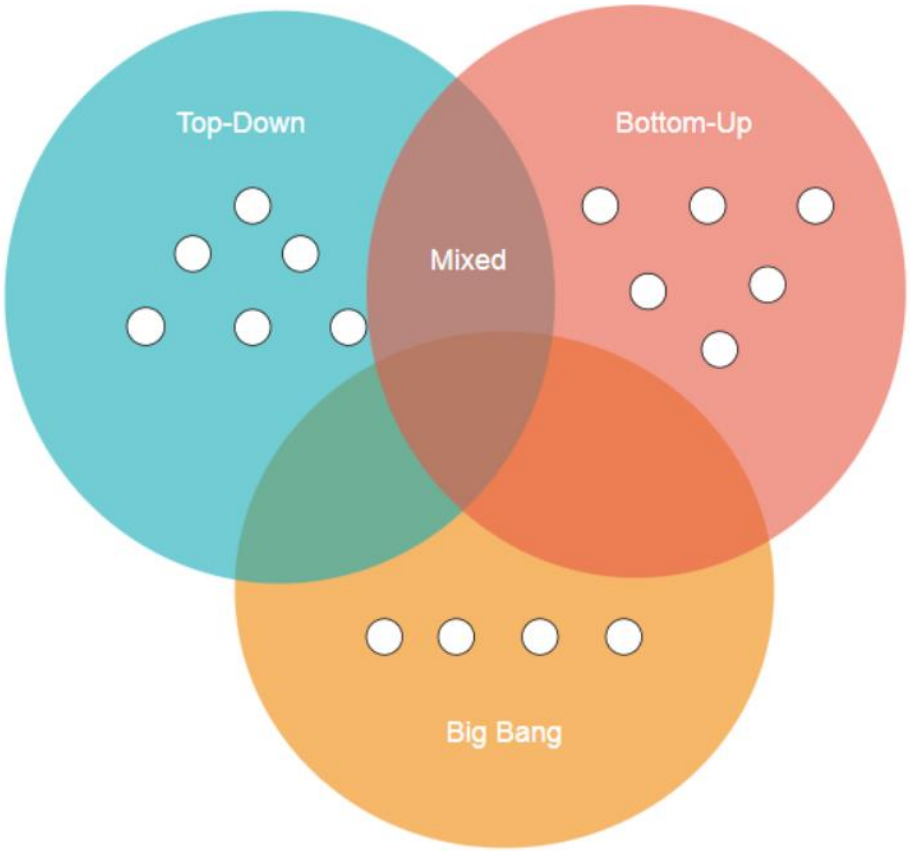

Рисунок 1.2 – Основні підходи інтеграційного тестування

Системне тестування – пов’язане з перевіркою необхідних операцій програмного забезпечення та його сумісності з операційними системами.

Іншими словами, ми тестуємо як технічні особливості, так і бізнес-логіку програмного забезпечення; ми запускаємо функціональні тести, щоб перевірити, що роблять різні функції системи, і нефункціональні тести, щоб перевірити, як ці функції працюють.

Наприклад, під час функціонального тестування ми перевіряємо, чи реагує функція входу, коли користувач вводить пароль. Але в нефункціональному тестуванні ми перевіряємо, скільки часу потрібно користувачеві для входу після введення пароля.

Щоб зробити це ефективно, його потрібно перевірити зовнішніми тестувальниками програмного забезпечення, які не мають абсолютно ніякого відношення до розробки програмного забезпечення. Тестування системи допомагає визначити, наскільки програмне забезпечення сумісне з операційними системами, для яких воно розроблено. Таким чином, усі готові версії програмного забезпечення випробовуються на всіх можливих операційних системах, які можуть мати користувачі. Бувають випадки, коли версія програми для Android може працювати належним чином, але версія для iOS може мати проблеми.

Системне тестування виявляє такі помилки, і розробники можуть виявити та виправити їх до запуску. Тестування системи, крім того, визначає, наскільки якісним є дизайн програмного забезпечення. Внутрішній код може працювати належним чином, але зовнішній дизайн може мати деякі робочі проблеми. Ось чому тестування системи є важливим. Таким чином, усі аномалії конструкції можна виявити та виправити.

Загальна поведінка програмного забезпечення також є ще одним аспектом, який тестування системи допомагає перевірити. Програмне забезпечення може бути належним чином розроблено, але все ж працювати несправно через різні

проблеми системи.

Тестування системи допомагає виявити будь-яку ненормальну поведінку та допомагає організаціям окреслити найкращі специфікації системи програмного забезпечення. Недотримання цих вимог може призвести до розповзання обсягу. Це також означає, що проект може накопичувати більше витрат, більшу тривалість і потребу в додаткових ресурсах. Система також не виправдає своїх обіцянок щодо надання бізнес-цінності для вашої організації.

Виявлені специфікації системи потім передаються кінцевим користувачам як вимоги, яким їхні пристрої повинні відповідати або перевищувати. Сьогодні кожне автентичне програмне забезпечення, яке ви купуєте, має мінімальні вимоги до операційної системи.

Приймальне тестування – рівень тестування програмного забезпечення подібний до тестування системи, але тут тестування виконується деякими обраними кінцевими користувачами. Це єдиний етап тестування програмного забезпечення, який здійснюють користувачі. Цей етап визначає, чи програмне забезпечення нарешті готове до запуску для широкого загалу. Усі вибрані користувачі висловлюють різні думки щодо роботи програмного забезпечення; вони повідомляють організації про те, чи відповідає програмне забезпечення їх різноманітним вимогам, і рекомендують області, які, можливо, потребують вдосконалення. Приймальне тестування також може називатися тестуванням прийнятності користувача.

Хоча кожна організація може проводити приймальне тестування з урахуванням власних особливостей – кого і де тестувати – життєво важливо, щоб буди вибирані користувачі з різними демографічними показниками, операційними системами та унікальним минулим досвідом використання подібного програмного забезпечення.

1.2.2 Методи тестування

Тестові кейси розробляються з використанням різних методів тестування для досягнення більш ефективного тестування. Цим забезпечується повнота програмного забезпечення та вибираються умови тестування, які отримують найбільшу ймовірність виявлення помилок. Отже, тестувальники не здогадуються, які тестові кейси обрати, а методи тестування дозволяють систематично розробляти умови тестування. Крім того, якщо комбінувати всі види існуючих методів тестування, можна отримати кращі результати, а якщо використовувати лише одну техніку тестування, можливий пропуск критичних помилок у розробленому програмному забезпеченні [2].

Програмне забезпечення можна перевірити двома способами. Іншими словами, можна виділити два різні методи:

– тестування чорного ящику;

– тестування білого ящику.

На рисунку 1.3 зображена схема тестування білого ящика – це техніка тестування, за якої перевіряються внутрішня структура, дизайн і кодування програмного забезпечення, щоб перевірити потік введення-виведення та покращити дизайн, зручність використання та безпеку.

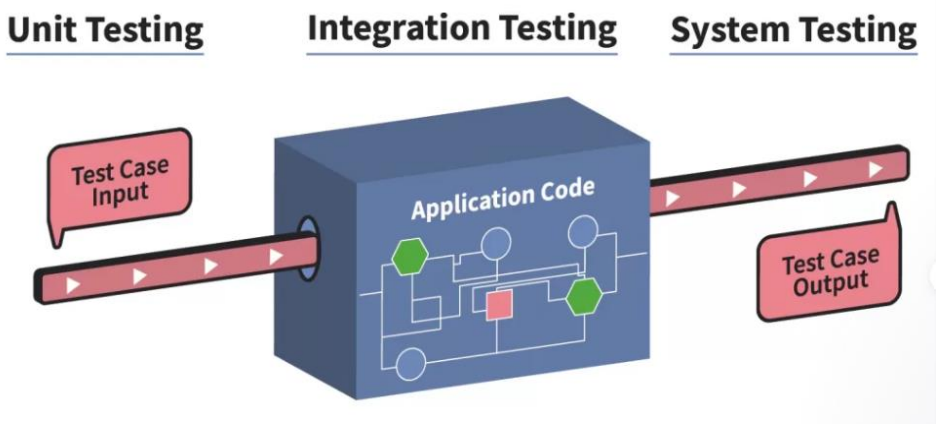

Рисунок 1.3 – Схема тестування методом білого ящика

У тестуванні білого ящика код видимий тестувальникам, тому його також називають тестуванням чистого ящика, тестування відкритого ящика, тестування прозорого ящика, тестування на основі коду та тестування скляного ящика.

Це одна з двох частин підходу Box Testing до тестування програмного забезпечення. Його аналог, тестування Blackbox, передбачає тестування з точки зору зовнішнього або кінцевого користувача. З іншого боку, тестування білої скриньки в розробці програмного забезпечення базується на внутрішній роботі програми та обертається навколо внутрішнього тестування. Термін «WhiteBox» використовувався через концепцію прозорої коробки. Прозора коробка або назва WhiteBox символізує можливість бачити крізь зовнішню оболонку (або «коробку») програмного забезпечення його внутрішню роботу. Подібним чином «чорний ящик» у «Тестуванні чорного ящика» символізує відсутність можливості побачити внутрішню роботу програмного забезпечення, тому можна перевірити лише роботу кінцевого користувача [2].

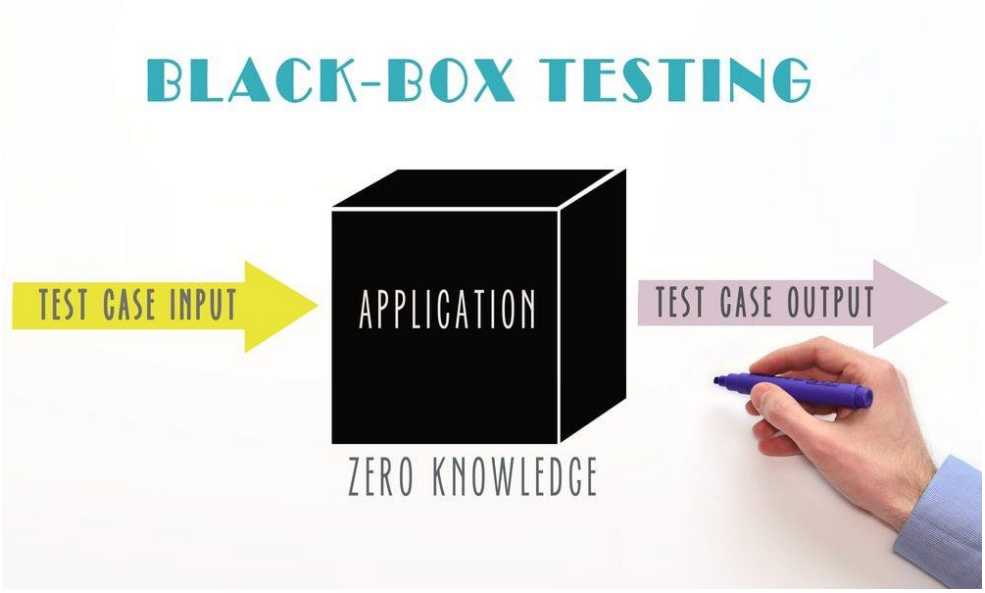

Рисунок 1.4 – Схема тестування методом білого ящика

На рисунку 1.4 зображена схема тестування чорного ящика яка передбачає тестування системи без попереднього знання її внутрішньої роботи. Тестер надає

вхідні дані та спостерігає за результатами, згенерованими системою, що тестується. Це дає змогу визначити, як система реагує на очікувані та несподівані дії користувача, час відповіді, проблеми зручності та надійності. Тестування «чорної скриньки» є потужною технікою тестування, оскільки вона наскрізно перевіряє систему. Подібно до того, як кінцевим користувачам «байдуже», як система закодована або сконструйована, і вони очікують отримати відповідну відповідь на свої запити, тестер може моделювати діяльність користувача та перевіряти, чи виконує система свої обіцянки. Попутно тест чорної скриньки оцінює всі відповідні підсистеми, включаючи UI/UX, вебсервер або сервер додатків, базу даних, залежності та інтегровані системи [2].

1.2.3 Види тестування

Під час процесу тестування існує величезна кількість типів тестів, які можна виконати, і кожен з них має свою мету. У цьому підрозділі будуть загалом описані всі значні види тестування, такі як:

– альфа-тестування;
– бета-тестування;
– одночасне тестування;
– стрес-тестування;
– тестування інсталяції;
– регресійне тестування;
– димове тестування.

Альфа-тестування проводиться вибраними користувачами разом із внутрішньою командою розробників, які контролюють середовище тестування та намагаються імітувати реалістичні умови під час тестування. Тут вибраним користувачам надаються паролі або ключі доступу, які дозволять їм входити та

виходити з платформи розробника без будь-яких перешкод для використання та тестування розробленого програмного забезпечення/програми. Під час альфа-тестування незначні помилки все ще можна знайти та виправити.

Бета-тестування проводиться вибраними користувачами на власних пристроях і операційних системах. Тут готове програмне забезпечення надсилається вибраним користувачам для використання та тестування протягом деякого часу, перш ніж надіслати різні відгуки. Очікується, що на рівні бета-версії програмне забезпечення не матиме дефектів, працюватиме ідеально та відповідатиме потребам користувачів.

Одночасне тестування – відстежує продуктивність та забезпечує результат тестованого програмного забезпечення під час його звичайної діяльності. Одночасне тестування виконується одночасно з функціональним тестуванням і його метою є визначення стабільності та продуктивності за очікуваних обставин.

Стрес-тестування – це тип тестування програмного забезпечення, який перевіряє стабільність і надійність програмного забезпечення. Метою стрес-тестування є вимірювання надійності програмного забезпечення та здатності до обробки помилок за умов надзвичайно високого навантаження та гарантування того, що програмне забезпечення не аварійно завершує роботу в критичних ситуаціях. Він навіть перевіряє межі нормальних робочих точок і оцінює роботу програмного забезпечення в екстремальних умовах.

Тестування інсталяції – тестування процедур для досягнення встановленої системи програмного забезпечення, яке можна використовувати, відоме як тестування інсталяції. У це тестування інсталяції включено перевірку повних або часткових оновлень та інсталяції/видалення інших функцій. Тестування встановлення гарантує, що програмне забезпечення було успішно встановлено з усіма властивими функціями чи ні. Це також називається тестуванням реалізації, в основному воно виконується на кінцевій фазі.

Регресійне тестування визначається як тип тестування програмного

забезпечення для підтвердження того, що нещодавня зміна програми чи коду не вплинула негативно на наявні функції. Регресійне тестування – це не що інше, як повний або частковий вибір уже виконаних тестів, які виконуються повторно, щоб переконатися, що існуючі функції працюють нормально. Це тестування проводиться, щоб переконатися, що нові зміни коду не мають побічних ефектів на існуючі функції. Це гарантує, що старий код продовжує працювати після внесення останніх змін у код.

Димове тестування – це процес тестування програмного забезпечення, який визначає, чи є розгорнута збірка програмного забезпечення стабільною чи ні. Димове тестування є підтвердженням для команди QA продовжити подальше тестування програмного забезпечення. Він складається з мінімального набору тестів, які виконуються на кожній збірці для перевірки функцій програмного забезпечення. Димове тестування також відоме як «Тестування верифікації збірки» або «Тестування достовірності». Говорячи простою мовою, димові тести означають перевірку того, що важливі функції працюють і в збірці, яка тестується, немає жодних перешкод. Це міні-швидкий регресійний тест основних функцій. Це простий тест, який показує, що продукт готовий до тестування. Це допомагає визначити, чи збірка має недоліки, щоб зробити подальше тестування марною тратою часу та ресурсів.

1.3 Тестування Web-додатків

Web-тестування або тестування web-сайту перевіряє вашу web-програму чи web-сайт на потенційні помилки перед тим, як він буде опублікований і доступний для широкого загалу. Wer-тестування перевіряє функціональність, зручність використання, безпеку, сумісність, продуктивність веб-програми або web-сайту. На цьому етапі перевіряються такі питання, як безпека веб-додатків,

функціонування сайту, його доступ для інвалідів, а також звичайних користувачів і його здатність обробляти трафік.

1.3.1 Тестування функціональності сайту

Тестування функціональності веб-сайту – це процес, який включає кілька параметрів тестування: інтерфейс користувача, API, тестування бази даних, тестування безпеки, тестування клієнта та сервера та основні функції веб-сайту. Функціональне тестування є дуже зручним і дозволяє користувачам виконувати як ручне, так і автоматичне тестування. Це виконується для перевірки функціональності кожної функції веб-сайту.

Web-тестування включає в себе:

- перевірка всіх посилань;
- перевірка форм;
- перевірка cookie;
- перевірка HTML та CSS;
- перевірка бізнес логіки.

Потрібно перевірити, чи всі посилання на web-сторінках працюють належним чином, і переконатися, що немає пошкоджених посилань. Посилання для перевірки включатимуть:

- вихідні посилання;
- внутрішні посилання;
- якірні посилання;
- посилання MailTo.

Необхідно переконатись що форми працюють належним чином. Перевірка включатиме:

– перевірки сценаріїв у формі працюють належним чином: наприклад, якщо користувач не заповнює обов’язкове поле у формі, чи відображається повідомлення про помилку;

– перевірити чи заповнюються значення за замовчуванням;

– після надсилання даних у формах, вони надсилаються до актуальної бази даних або прив’язуються до робочої електронної адреси;

– форми оптимально відформатовані для кращої читабельності.

Треба перевірити що cookie працюють належним чином. Файли cookie – це невеликі файли, які web-сайти використовують для запам’ятовування активних сеансів користувача, тому вам не потрібно входити щоразу, коли ви відвідуєте веб-сайт. Тестування файлів cookie включатиме:

– тестові файли cookie (сеанси) видаляються або після очищення кешу, або після закінчення терміну їх дії;

– видаліть файли cookie (сесії) і перевірте, чи запитуються облікові дані для входу під час наступного відвідування сайту.

Перевірка HTML та CSS виконується для того, щоб переконатися, що пошукові системи можуть легко сканувати сайт. Це включатиме:

– перевірка синтаксичних помилок;

– читабельні схеми кольорів;

Відповідність стандартам. Потрібно переконатися, що дотримуються такі стандарти, як W3C, OASIS, IETF, ISO, ECMA або WS-I.

Перевірка бізнес логіки буде включати:

– тестування наскрізного робочого процесу/бізнес-сценаріїв, які проводять користувача через низку веб-сторінок;

– також перевіряються негативні сценарії, щоб переконатися, що коли користувач виконує неочікуваний крок, web-програма відображала відповідне повідомлення про помилку.

1.3.2 Тестування зручності використання

Тестування зручності використання, також відоме як тестування взаємодії з користувачем – це метод тестування для визначення того, наскільки простим і зручним є web-застосунок. Невелика група цільових кінцевих користувачів використовує програмне забезпечення для виявлення дефектів зручності використання. Тестування зручності використання головним чином зосереджується на зручності користувача у використанні програми, гнучкості програми та здатності програми відповідати поставленим цілям.

Перевірка навігації сайту: меню, кнопки або посилання на різні сторінки вашого сайту мають бути легко видимими та узгодженими на всіх web-сторінках.

Перевірка вмісту:

– вміст має бути розбірливим, без орфографічних чи граматичних помилок;

– зображення, якщо вони є, мають містити текст «alt».

1.3.3 Тестування інтерфейсу

Тут потрібно перевірити три області: додаток, веб сервер і сервер бази даних:

1) додаток: тестові запити правильно надсилаються до бази даних, а вихід на стороні клієнта відображається правильно; помилки, якщо такі є, мають виявлятися програмою та показуватися лише адміністратору, а не кінцевому користувачеві;

2) вебсервер: тестовий веб-сервер обробляє всі запити програм без будь-якої відмови в обслуговуванні;

3) сервер бази даних: необхідно переконатися, що запити, надіслані до

бази даних, дають очікувані результати.

Перевірте реакцію системи, коли зв’язок між трьома рівнями (програма, Інтернет і база даних) не може бути встановлено, то на стороні користувача має відобразитися відповідне повідомлення.

1.3.4 Тестування бази даних

База даних є одним із критично важливих компонентів web-програми, і її необхідно ретельно протестувати. Тестування включатиме:

– перевірити, чи не відображаються помилки під час виконання запитів;

– цілісність даних підтримується під час створення, оновлення або видалення даних у базі даних;

– тестові дані, отримані з бази даних, точно відображаються у web-програмі;

– перевірити час відповіді на запити.

1.3.5 Тестування на сумісність

Тести на сумісність гарантують, що ваш веб-додаток правильно відображається на різних пристроях. Тест на сумісність браузера: той самий web-сайт у різних браузерах відображатиметься по-різному. Потрібно перевірити, чи правильно web-програма відображається в усіх браузерах, чи добре працює JavaScript, AJAX і автентифікація. Також потрібно перевірити сумісність мобільного браузера. Візуалізація web-елементів, таких як кнопки, текстові поля тощо, змінюється зі зміною операційної системи. Слід переконатися, що web-сайт добре працює для різних комбінацій операційних систем, таких як Windows,

Linux, Mac, і таких браузерів, як Firefox, Internet Explorer, Safari тощо.

1.3.6 Тестування безпеки

Тестування безпеки є життєво важливим для веб-сайтів електронної комерції, які зберігають конфіденційну інформацію про клієнтів, як-от кредитні картки. Тестування включатиме:

– перевірити що не авторизований користувач не має доступу до захищених сторінок;

– заборонені типи файлів не можуть бути завантажені без відповідного доступу;

– переконатися в тому що сессія користувача автотично завершується після тривалої бездіяльності користувача;

– у разі використання сертифікатів SSL web-сайт має перенаправляти на зашифровані сторінки SSL.

1.4 Висновки до першого розділу

В першому розділі наведена загальна теоретична інформація про тестування. Наведені приклади рівнів тестування, методів тестування ти видів тествання. При розробці автотестів було використано метод чорної скриньки, тому що я не мав доступу до коду. Також було проаналізовано що необхідно тестувати у веб-додатках.

2 ІНСТРУМЕНТИ ДЛЯ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ WEB-ДОДАТКІВ

2.1 Критерії вибору інструментів

Вибір правильного інструменту може бути складним завданням. Наступний критерій допоможе обрати найкращий інструмент для автоматизованого тестування:

– простота використання;

– тестування бази даних;

– тестування зображень;

– підтримка різних типів тестів;

– підтримка декількох фреймворків тестування;

– ідентифікація об’єкта;

– легко налагоджувати сценарії програмного забезпечення для автоматизації;

– звіти та результати випробувань;

– мінімізована вартість навчання вибраних інструментів.

Вибір інструменту – одна з найбільших проблем, яку потрібно вирішити, перш ніж розпочинати розробку автотестів.

2.2 Найбільш популярні інструменти для автоматизації тестування

Типова система автоматизації забезпечує середовище для виконання планів тестування та генерування звітів. Це спеціалізовані інструменти, які допомагають

виконувати повсякденні завдання автоматизації тестування. Незалежно від того, чи це програма для виконання тестів, інструмент для запису дій чи інструмент для web-тестування, він створений для того, щоб усунути всю важку роботу зі створення тестових сценаріїв і залишити вам більше часу для перевірки якості. Автоматизація тестування – це перевірений економічно ефективний підхід до вдосконалення розробки програмного забезпечення. Тому вибір найкращої системи автоматизації тестування може виявитися вирішальним для результатів тестування та термінів контролю якості [3].

Згідно з опитуванням, майже 37 % організацій зараз інтегрують автоматизацію тестування у свій процес безперервної інтеграції та безперервної доставки (CI/CD), що дозволяє розробникам миттєво знати, чи порушила остання зміна коду будь-який аспект програми, після його розгортання [3].

Оскільки попит на автоматизоване тестування постійно зростає, не дивно, що засобів багато. Було відібрано найкращі з них, доступних у 2022 році (без певного порядку):

- Selenium;
- Cypress;
- WebDriverIO;
- Robot Framework;
- EarlGrey;
- Appium.

2.3 Загальна інформація про Cypress

Cypress – перспективна платформа веб-тестування. Цей засоб для автоматизації веб-тестування є надійним доповненням до Selenium, який довгий

час був де-факто фреймворком для веб-автоматизації. Cypress – це інструмент наскрізного тестування на основі JavaScript, призначений для автоматизації сучасних веб-тестів. Цей інструмент, зручний для розробників, працює безпосередньо у браузері за допомогою техніки маніпулювання DOM і дозволяє розробникам інтерфейсу та інженерам із забезпечення якості писати автоматизовані тести, усуваючи при цьому проблемні точки [4].

У порівнянні з іншими фреймворками автоматизації Cypress більш універсальний, оскільки він написаний на JavaScript і заснований на Mocha та Chai. Він також використовує Node.js під час роботи в браузерах. Той факт, що Cypress заснований на JavaScript, є надзвичайно важливим, оскільки JavaScript є основною мовою, яку використовують розробники для розробки веб-сайтів. Оскільки тести Cypress створюються мовою, яка вже знайома розробникам, це робить інструмент набагато зручнішим для користувача. За допомогою Cypress також можливо запускати кросбраузерне тестування. Ви можете виконувати тести за допомогою Cypress у Firefox і браузерах сімейства Chrome, таких як Edge і Electron [4].

Легко розпочати роботу з автоматизацією Cypress для веб-тестування. Якщо ви працювали з Selenium, то знаєте, що перед початком тестування потрібно вибрати всі потрібні вам залежності та бібліотеки. У Cypress ці залежності та бібліотеки вже налаштовані без. Крім того, Cypress поставляється в комплекті з браузером Chrome, тому немає складного середовища для налаштування. Крім того, ви можете протестувати за допомогою Cypress будь-який інший браузер, встановлений на вашій локальній машині. Ця гнучкість і просте налаштування є перевагами порівняно з Selenium, який вимагає від користувача завантажити відповідний драйвер і налаштувати сітку для початку тестування [4].

Ви можете швидко й легко налагоджувати свої веб-програми за допомогою Cypress. Якщо тести не повертають помилки, ви отримуєте пропозиції щодо

усунення дефекту. Звідти ви можете налагоджувати безпосередньо з Chrome DevTools. Крім того, Cypress підтримує такі можливості, як подорож у часі та перезавантаження в реальному часі, тож розробники можуть перевіряти код свого веб-сайту під час і після виконання тесту. Оскільки Cypress також має доступ до кожного об’єкта, це спрощує аналіз помилок. Cypress може надавати знімки екрана помилок тестування, що робить пошук дефектів і налагодження програм швидким і простим [4].

Cypress відомий своїм швидким виконанням тестів – із часом відповіді менше 20 мс. Cypress має автоматичне очікування, вбудоване в структуру, що означає, що вам не потрібно визначати неявні та явні очікування. Фреймворк автоматично очікує на завантаження DOM, анімацію, елементи тощо. Крім того, фреймворк також запускає наступні тести автоматично після виконання першого. Це виключає час простою та необхідність вручну запускати наступний тест [4].

Cypress – це безкоштовний фреймворк із відкритим кодом. Він працює за моделлю freemium, де ви можете використовувати безкоштовну або платну версію. Платна версія включає розширені функції, такі як інформаційна панель з артефактами, такими як знімки DOM, які корисні для налагодження, а також зберігання відео. Інтерес до Cypress зростає, і він може похвалитися активною спільнотою на GitHub, Gitter і StackOverflow. Крім того, Cypress пропонує надійну документацію [4].

2.4 Загальна інформація про Selenium

Selenium – це один із найпоширеніших засобів автоматизованого тестування веб-інтерфейсу користувача з відкритим кодом. Його спочатку розробив Джейсон Хаггінс у 2004 році як внутрішній інструмент Thought Works.

Selenium підтримує автоматизацію в різних браузерах, платформах і мовах програмування. Selenium можна легко розгорнути на таких платформах, як Windows, Linux, Solaris і Macintosh. Крім того, він підтримує ОС (операційну систему) для мобільних додатків, таких як iOS, windows mobile та android.

Selenium підтримує різноманітні мови програмування завдяки використанню драйверів, специфічних для кожної мови. Selenium підтримує такі мови, як C#, Java, Perl, PHP, Python і Ruby. Наразі веб-драйвер Selenium найбільш популярний для Java та C#. Тестові сценарії Selenium можна закодувати будь-якою з підтримуваних мов програмування та запускати безпосередньо в більшості сучасних веб-браузерів. Браузери, які підтримує Selenium, включають Internet Explorer, Mozilla Firefox, Google Chrome і Safari.

Більшість програмістів, які створюють програми для веб-сайтів і бажають час від часу тестувати їх, використовують Selenium. Однією з найбільших переваг Selenium, яка зробила його таким популярним, є його гнучкість. Будь-хто може використовувати Selenium для тестування коду та програм. Крім того, професіонали можуть налагоджувати та виконувати візуальні регресійні тести відповідно до вимог веб-сайту чи коду.

2.5 Порівняльна характеристика Cypress та Selenium

Кожен прикладний веб-проект має свої вимоги до мови та інструментів. Це єдине, що не змінюється протягом усього проекту. Вибір інструменту тестування, який підтримує кілька мов, це добре, але це має бути найкращий інструмент для вашої мови програмування [5].

Що, якщо ви працюєте з інструментом, який чудово підходить для тестування, але потребує багато наполегливої роботи та часу, щоб адаптувати його до вашого випадку використання, то цей інструмент може бути не вдалим

вибором. Ключовим є вибір інструменту, який ефективно виконує роботу, не вимагаючи тривалого навчання [5].

Робота над веб-проектом великого масштабу вимагає часу. Природно, що програмісти більше зосереджуються на розробці і хочуть витрачати менше часу на тестування. Але витрачання меньшогу часу на тестування може призвести до неякісного тестування і, зрештою, до провалу проєкту. Отже, потрібен інструмент, який міг би виконувати роботу ефективно та за найменший час [5].

В таблиці 2.1 розглянуті основні порівняльні характеристики Cypress та Selenium.

Таблиця 2.1 – Порівняння Cypress та Selenium

Порівняльні характеристики	Cypress	Selenium
Види тестування	Тестування інтерфейсу користувача, тестування API, наскрізне тестування (end-to-end)	Наскрізне тестування (end-to-end), тестування API не підтримується
Простота використання	Якщо ви знайомі з JavaScript, це буде легко. В іншому випадку буде трохи складно. Це зручно для розробників, оскільки розроблено з урахуванням розробників.	Оскільки Selenium підтримує кілька мов, люди можуть швидко почати писати тести, але це забирає більше часу, ніж Cypress, оскільки вам потрібно вивчити певний синтаксис.

Продовження таблиці 2.1

Порівняльні характеристики	Cypress	Selenium
Швидкість	Cypress має іншу архітектуру, яка не використовує веб-драйвер і тому є швидшим.	Через його архітектуру важко створювати прості швидкі тести. Однак сама платформа швидка, і ви можете запускати багато тестів у масштабі, паралельно та між браузерами.
Підтримувані мови програмування	JavaScript/Typescript	Підтримуються кілька мов, наприклад Java, JavaScript, Perl, PHP, Python, Ruby, C# тощо
Простота налаштування	Просто запустіть таку команду: npm install Cypress –save-dev Cypress не вимагає встановлення іншого компонента (веб-драйвера), як це робить Selenium.	Оскільки Selenium має два компоненти прив’язки та веб-драйвер. Налаштування є більш складний і трудомісткий.
Документація	Хороші зразки коду та документація.	Середня документація.
Підтримувані браузери	Усі браузери на основі хрому (Chrome, Edge, Brave) і Firefox.	Усі браузери: Chrome, Opera, Firefox, Edge, Internet Explorer тощо.

Selenium – це доволі стара система, яка існує вже давно, з 2002 року. Selenium має величезну базу встановлення та використовується ширше, ніж будь-який інший інструмент автоматизованого тестування. Cypress є новим і набуває популярності. Cypress є швидким для API, інтерфейсного та наскрізного тестування завдяки своїм архітектурним відмінностям. Недоліком є те, що поки що він підтримує лише JavaScript [5].

Незважаючи на те, що Selenium підтримує кілька мов, розробникам і тестувальникам важко швидко адаптуватися, оскільки він складний. Cypress, тим часом, легше освоїти і швидко використовувати. Через його простішу архітектуру тестувальники, як правило, віддають перевагу йому для автоматизованого тестування. Як наслідок, Cypress є переможцем за простоту використання та швидше виконання, але він не може повністю замінити Selenium, а також не може використовуватися разом із Selenium. Selenium також підтримує найбільші проекти, запускаючи тисячі тестів [5].

2.6 Розробка діаграм

Одним засобом концептуального моделювання даних є універсальна мова моделювання UML. UML надає можливість для створення візуальних моделей, які одноманітно розуміються усіма розробниками, залученими до проекту, і є засобом комунікації в рамках проекту. Діаграма в UML – це графічне представлення набору елементів. Діаграми малюють для візуалізації системи з різних точок зору.

Діаграма послідовності призначена для відображення часових залежностей, що виникають в процесі спілкування між об'єктами. На діаграмі послідовності зображаються тільки ті об'єкти, які безпосередньо беруть участь у взаємодії і не показуються можливі статичні асоціації з іншими об'єктами. Для

діаграми ключовим моментом є саме динаміка взаємодії об'єктів в часі. При цьому діаграма послідовності має як би два виміри. Одне – зліва направо у вигляді вертикальних ліній, кожна з яких зображує лінію життя окремого об'єкта, який бере участь у взаємодії. Графічно кожен об'єкт зображується прямокутником і розташовується у верхній частині своєї лінії життя. Усередині прямокутника записуються ім'я об'єкта і ім'я класу, розділені двокрапкою.

На рисунку 2.1 зображена діаграма послідовності, яка описує створення та обробку замовлення. Спочатку користувач переходить на головну сторінку сайту. Після цього користувач переглядає товари які зображені на сторінці. Потім він додає товари до кошику. Після додавання товарів до кошику користувач їх переглядає. Як що він додав вірні товари, користувач переходить на сторінку оформлення замовлення. Потім копистувач заповнює форму і як що, всі введені дані вірні, то виконується оформлення замовлення.

На рисунку 2.2 зображена програмна реалізація оформлення замовлення яке було описано у вигляді діаграми на рисунку 2.1. Спочатку викликаються дві функції AddProductToCart() із числовим параметром. Ця фунція шукає продукт по індексу який передається у параметрі та додає продукт до кошику.

Далі викликаються два методи visitPage() та clickCheckoutButton(), класу shoppingCartPage. Метод visitPage() відкриває сторінку Shopping cart, метод clickCheckoutButton() натискає на кнопку Checkout.

Далі викликаються методи selectGuestCheckoutRadioButton(), clickContinueButton(), checkOrderSummary(), fillFirstNameInput(), fillLastNameInput(), fillEmailInput(), fillAddress1Input(), fillCityInput(), fillRegionSelect(), fillZipCodeInput(), clickContinueButton() класу guestCheckoutPage. Методи назви яких починаютбься з fill… відповідають за заповнення полів, метод clickContinueButton() натискає на кнопку Continue.

Останніми викликаються методи checkBreadcrumbs(), checkItemsInYourCart(), clickConfirmOrderButton() класу

checkoutConfirmationPage. Метод checkBreadcrumbs() перевіряє тест в блоку навігації. Метод checkItemsInYourCart() порівнює дані з таблички Items In YourCart з даними які зберігаються в кошику. Метод clickConfirmOrderButton() напистає на кнопку Confirm Order.

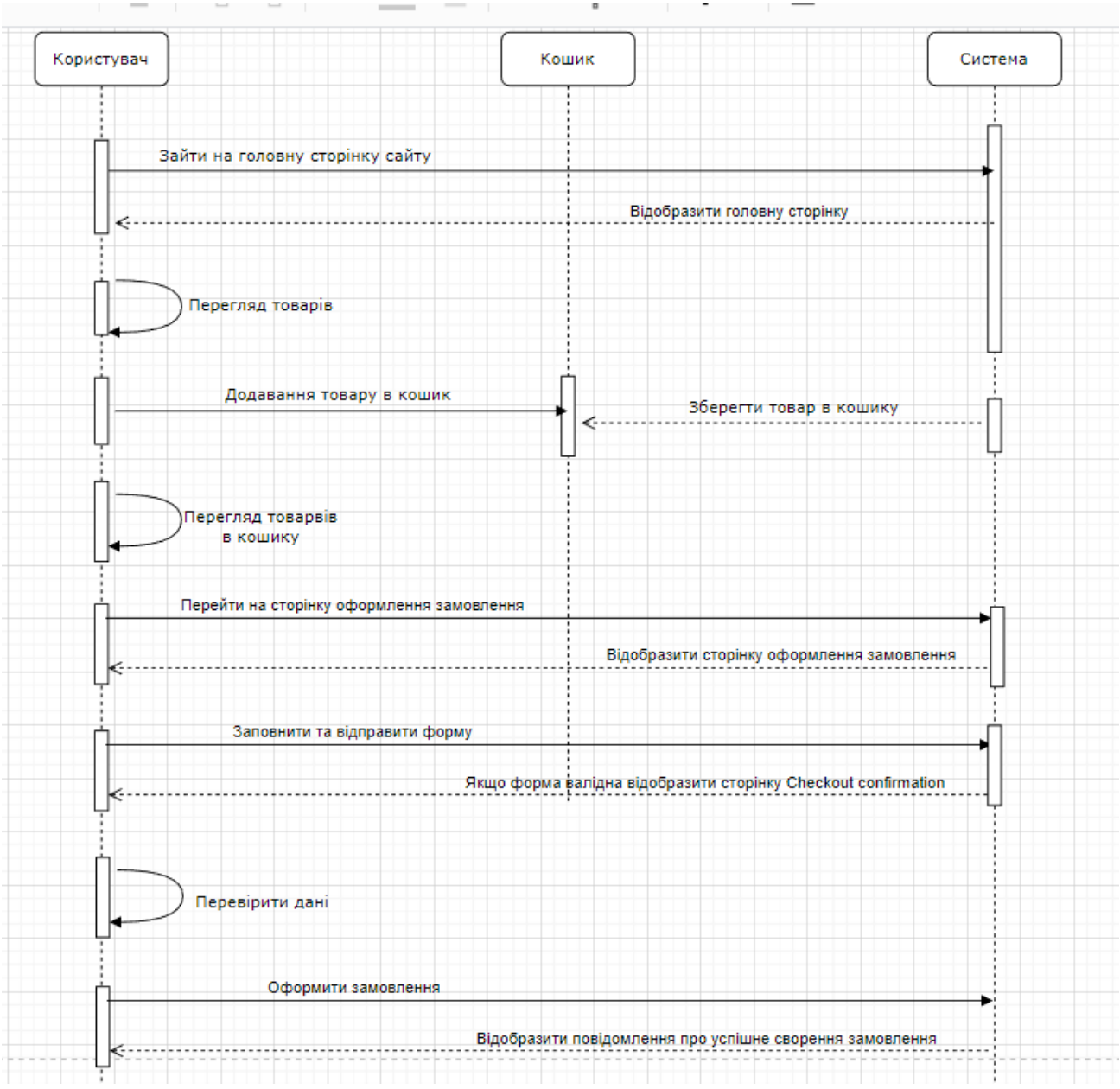

Рисунок 2.1 – Діаграма послідовності оформлення замовлення

```
describe('Checkout confirmation page testing', function(){
    it('Positive scenario', function(){
        cy.AddProductToCart(0)
        cy.AddProductToCart(4)

        shoppingCartPage
        .visitPage()
        .clickCheckoutButton()

        guestCheckoutPage
        .selectGuestCheckoutRadioButton()
        .clickContinueButton()
        .checkOrderSummary()
        .fillFirstNameInput(guestCheckoutPageData.firstName)
        .fillLastNameInput(guestCheckoutPageData.lastName)
        .fillEmailInput()
        .fillAddress1Input(guestCheckoutPageData.address1)
        .fillCityInput(guestCheckoutPageData.city)
        .fillRegionSelect(3)
        .fillZipCodeInput(guestCheckoutPageData.zipCode)
        .clickContinueButton()

        checkoutConfirmationPage
        .checkBreadcrumbs()
        .checkItemsInYourCart()
        .clickConfirmOrderButton()
    })
})
```

Рисунок 2.2 – Програмна реалізація оформлення замовлення

На рисунку 2.3 зображена діаграма послідовності, яка описує використання користувачем головної сторінки сайту. Спочатку користувач переходить на головну сторінку сайту. Після цього користувач переглядає товари які зображені

на сторінці. Котстувач переглядає категорії та під категорії. Користувач переглядає банер, після цього змінює валюту та додає випадковий продукт до кошику.

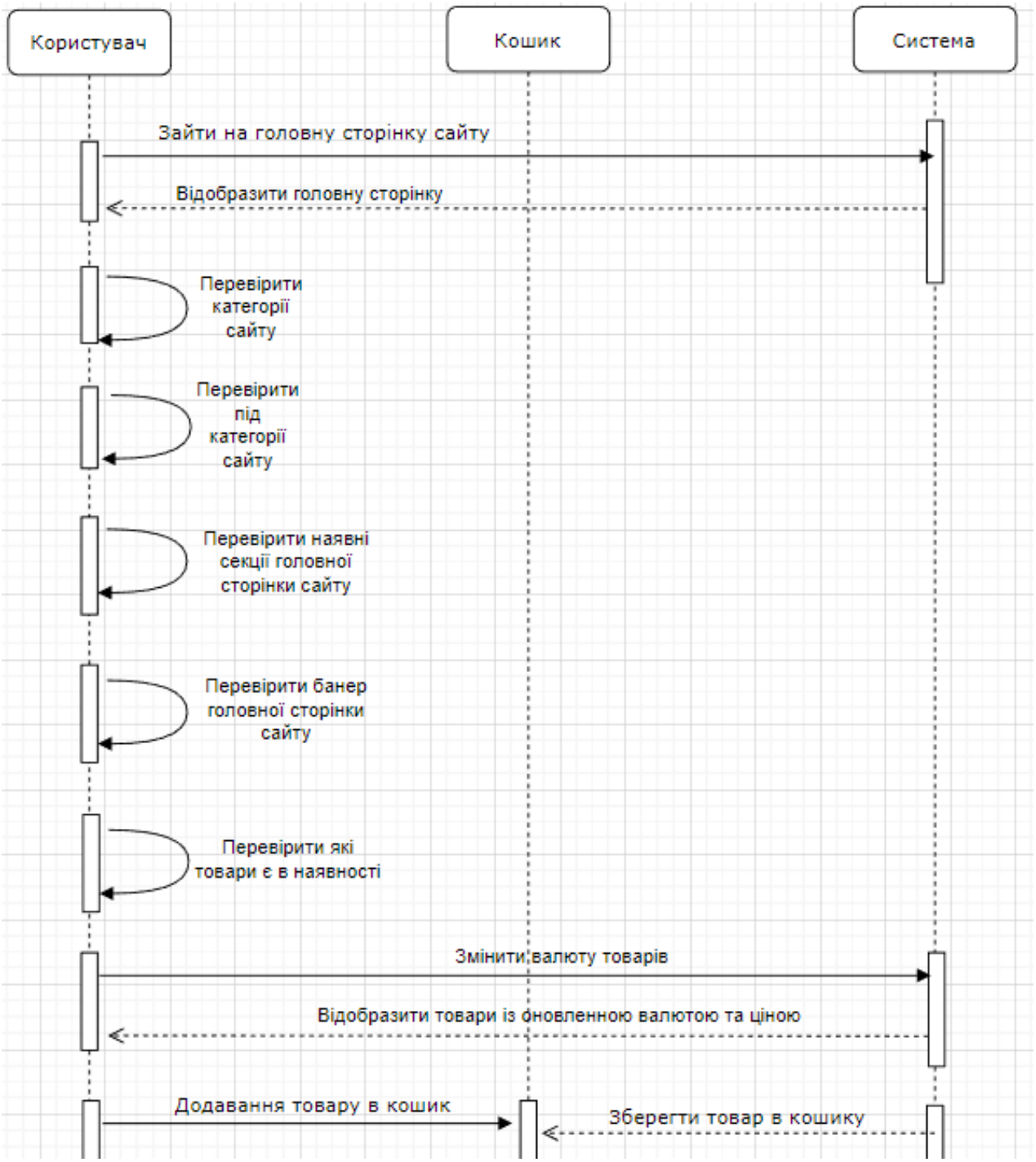

Рисунок 2.3 – Діаграма послідовності використання головної сторінки

На рисунку 2.4 зображена програмна реплізація використання головної сторінки яка була описана у вигляді діаграми на рисунку 2.3. Щоб відтворити дії користувача на голоній сторінці викликаються методи visitMainPage(), checkCategories(), checkSubcategories(), checkPromoSection(), checkSlide(), moveSlide(), checkPrice(), changeCurrency(), checkOutOfStockProducts(), addProductToChart() класу mainPage.

```
mainPage.visitMainPage()
    .checkCategories()
    .checkSubcategories()
    .checkPromoSection()
    .checkSlide(mainPage.mainShopPageLocators.bunner.slide1, mainPageData.banner.slide1)
    .moveSlide()
    .checkSlide(mainPage.mainShopPageLocators.bunner.slide2, mainPageData.banner.slide2)
    .moveSlide()
    .checkSlide(mainPage.mainShopPageLocators.bunner.slide3, mainPageData.banner.slide3)
    .checkPrice(mainPageData.featuredProducts)
    .checkPrice(mainPageData.latestProducts)
    .checkPrice(mainPageData.bestsellersProducts)
    .checkPrice(mainPageData.specialsProducts)
    .changeCurrency('Euro')
    .checkPrice(mainPageData.featuredProducts)
    .checkPrice(mainPageData.latestProducts)
    .checkPrice(mainPageData.bestsellersProducts)
    .checkPrice(mainPageData.specialsProducts)
    .changeCurrency('Sterling')
    .checkPrice(mainPageData.featuredProducts)
    .checkPrice(mainPageData.latestProducts)
    .checkPrice(mainPageData.bestsellersProducts)
    .checkPrice(mainPageData.specialsProducts)
    .changeCurrency('Dollar')
    .checkPrice(mainPageData.featuredProducts)
    .checkPrice(mainPageData.latestProducts)
    .checkPrice(mainPageData.bestsellersProducts)
    .checkPrice(mainPageData.specialsProducts)
    .checkOutOfStockProducts()
    .addProductToChart()
```

Рисунок 2.4 – Програмна реалізація використання головної сторінки

Метод visitMainPage() відкриває головну сторінку сайту. Метод checkCategories() перевіряє назви головних категорій. Метод checkSubcategories() перевіряє назви під категорій. Метод checkPromoSection() перевіряє контент в

промо секції. Метод checkSlide() перевіряє контент на слайдах головного банера. Метод moveSlide() змінює слайд головного банера. Медод checkPrice() приймає яа параметр назву секції па перевіряє ціну продуктів які знаходяться в переданій секції. Метод changeCurrency() приймає як памерт назву валюти та змінює її. Метод checkOutOfStockProducts() перевіряє чи правельні товари помічені як “немає в наявності”. Метод addProductToChart() додає випадкой товар з головної сторінки до кошика та перевіряє чи правильний товар відображається в кошику.

Діаграма прецедентів – це діаграма на якій зображено відношення між акторами та прецедентами в системі. Також, перекладається як діаграма віріантів використання. Суть даної діаграми полягає в наступному: проектована система представляється у вигляді безлічі сутностей чи акторів, що взаємодіять із системою за допомогою так званих варіантів використання. Діаграми прецедентів відображають елементи моделі варіантів використання. На рисунку 2.5 зображена діаграма прецедентів яка відображає взаємодії трьох актоpіа із системою інтернет магазину. В ролі акторів вистувають: Адміністратор, Користувач та Платіжна система.

Адміністратор пов'язаний з шістьма прецедентами:

- логін в систему;
- додавання, оновлення, видалення категорій;
- додавання, оновлення, видалення продуктів;
- додавання, оновлення, видалення користувачів;
- редагування, видалення замовлень;
- вихід із системи.

Користувач пов'язаний з сімома прецедентами:

- логін в систему;
- переглядпродуктів;
- додавання продукту в кошик;

– видалення продукту з кошику;

– оформлення замовлення;

– оплата замовлення;

– вихід із системи.

Платіжна система пов'язана лише з одним прецедентом – оплатою замовлення.

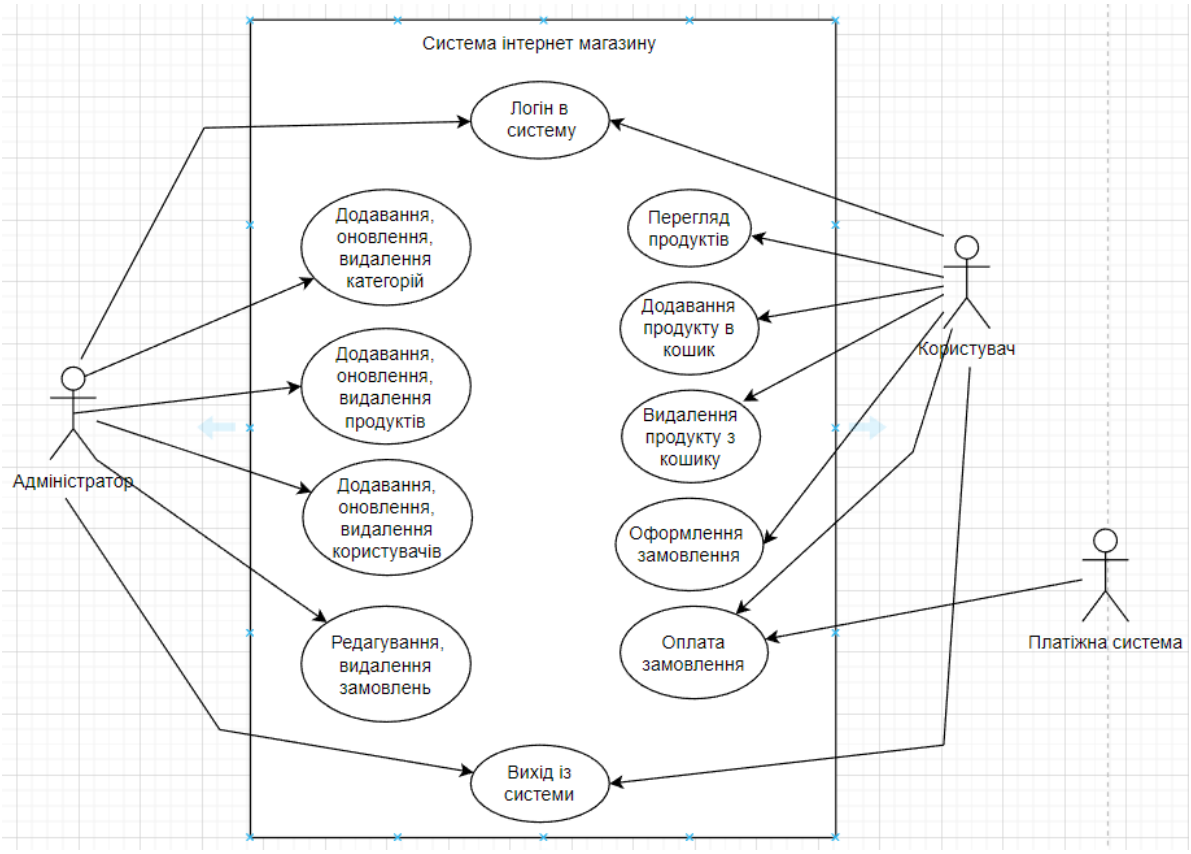

Рисунок 2.5 – Діаграма прецедентів

2.7 Висновки до другого розділу

Наразі в наш час є багато засобів автоматизації тестування. Кожен інструмент цікавий та унікальний. Було розглянуто основні засоби та їх

характеристика. Також було проалізовано переваги та недоліки найбільш популярних систем, Cypress і Selenium. Були проаналізовані основні критерії вибору системи автоматизації. Тому було прийнято рашення використовувати Cypress. Були розроблені діаграми прецедінтів та полідовності які описують можливості використання сайту який буде використаний для автоматезаціїї.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Опис web-додатку

Для автоматизованого тестування було обрано веб-додаток інтернет-магазину: https://automationteststore.com/. На рисунку 3.1 зображена головна сторінка сайту.

Automation Test Store – сайт, який містить основні функції інтернет-магазину. Використовуючи сайт, користувач може переглядати каталог товарів, шукати товари, дивитися опис товарів, додавати в кошик, оформляти заявку.

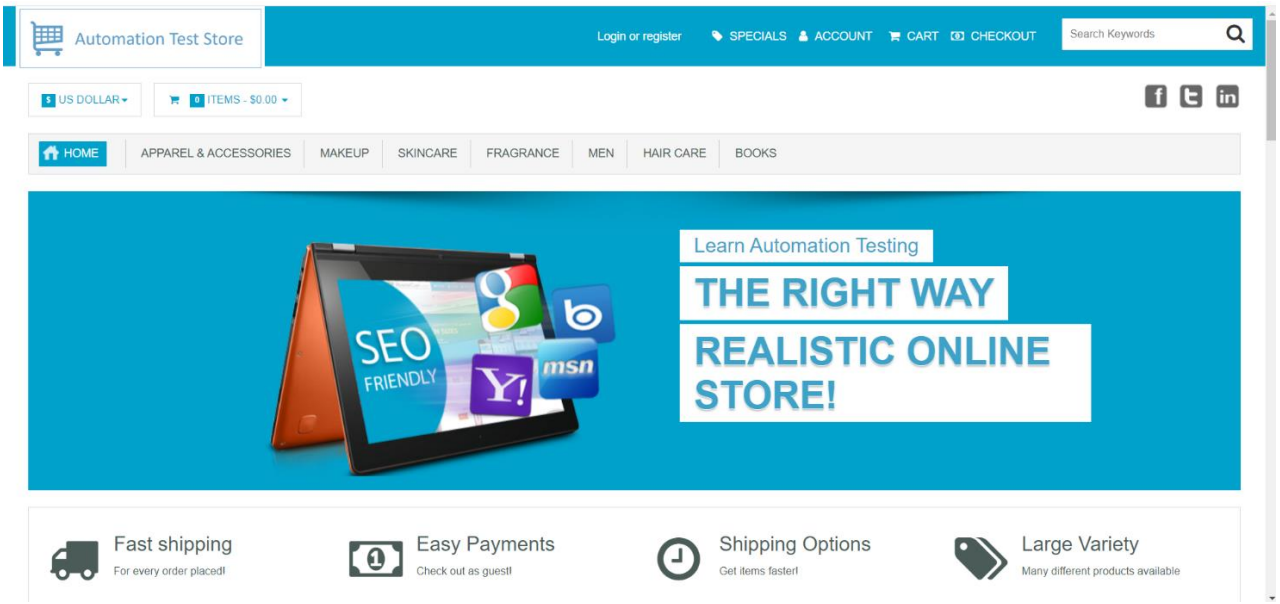

Рисунок 3.1 – Головна сторінка сайту Automation test store

В процесі тестування сайту буде використано ad-hoc тестування через відсутність специфікації, а також через обмеженість ресурсів на формалізацію тестів.

Планується чотири етапи проведення процесу тестування:

– перший етап полягає в складанні тест-плану і чек-листа, а також часткового прогону функціональних тестів;

– другий етап буде складатися з тестування верстки сайту;

– на третьому етапі буде проведений прогін авто тестів;

– четвертий етап полягає в тестуванні зручності використання продукту з описом пропозицій щодо поліпшення.

Таким чином, досягається максимальна деталізація глибини тестування, що, в свою чергу, дозволяє більш точно визначити витрачені ресурси, а також дозволяє розробникам проєкту виправляти дефекти на ранніх етапах.

3.2 Огляд розроблених класів

Клас *mainShopPage* описує головну сторінку сайту.

В класі розроблені наступні методи, в прикладі представлені не всі методи класу:

– visitMainPage();

– changeCurrency(currency);

– checkPrice();

– checkOutOfStockProducts();

– addProductToChart();

– checkSlide().

Розглянемо методи класу mainShopPage детальніше.

Метод visitMainPage(): цей метод відкриває головну сторінку сайту.

Метод changeCurrency(currency): цей метод приймає як параметр назву валюти, та змінює валюту для товарів на головній сторінці.

Метод checkPrice() приймає за параметр назву секції з головної сторінки, та

перевіряє ціни товарів які там відображаються.

Метод checkOutOfStockProducts() перевіряє чи коректні товари відображаються як “немає в наявності”, в усіх секціях головної сторінки.

Метод addProductToChart() додає один випадкой продукт з головної сотрінки до кошику, та перевіряє чи правильна інформація відображається у кошику після додавання товару. Також у методі присутня перевірка на продукти яких немає в наявності, вони будуть ігноруватися методом.

Метод checkSlide() приймає два параметри: селектор слайду (його css класс або айді), та контент якій має були присутнім на слайді. Переданий котент має співпадати з тим який був отриман зі сторінки.

Клас *shoppingCartPage* описує сторінку перегляду товарів які були додані до кошику.

В класі розроблені наступні методи:

– visitMainPage();

– checkCartTable();

– removeItemFromCart().

Розглянемо методи класу shoppingCartPage детальніше.

Метод visitMainPage() відкриває сторінку перегляду товарів які були додані до кошику.

Метод checkCartTable() перевіряє чи є якісь дані в табличці Shopping Cart та у кошику який відображається в хедері сторінки. Якщо ці єлементи не пусті, то метод отримує з них дані про товари які там відображаються, а потім порівнює їх між собою, дані повинні співпадати. Як що, кошик і таблиця пусті, то метод перевіряє чи відображається повідомлення про те що кошик пустий.

Метод removeItemFromCart() отримує дані з таблиці Shopping Cart та кошику. Як що, там зберігається лише один продук, то натискається кнопка яка видаляє цей продукт із кошику. Після видалення іде перевірка на те, чи відображається повідомлення про те, що кошик пустий, та чи відображаються

якісь дані у верхньому кошику. Як що, товарів в кошику більше ніж один, то видяляється перший із спискую. Далі іде перевірка на те що товарів стало на один меньше.

Клас *randomSubcategoryPage* описує сторінку підкатегорії сайту.

В класі розроблені наступні методи:

– selectRabdomSubcategoty();

– addToCartAllProducts().

Розглянемо методи класу randomSubcategoryPage детальніше.

Метод selectRabdomSubcategoty() отримує всі можливі підкатегорії сайту, потім випадковим чином вибирається одна підкатегорія. Після цього, відкривається сторінка з товарами обранної підкатегорії.

Метод addToCartAllProducts() знаходить всі продукти які відображаються на сторінці. Після цього виконується фільтрація тих продуктів які є в наявності. Після цього продукти додаються до кошику.

Клас *orderDetailsPage* описує сторінку пошуку та перегляду створеного замовлення.

В класі розроблені наступні методи:

– openOrderDetailsPage();

– checkFormValidation();

– checkOrderDetails();

– fillOrderIdInput();

– fillEmailInput().

Розглянемо методи класу orderDetailsPage детальніше.

Метод openOrderDetailsPage() відкриває сторінку для пошуку створеного замовлення.

Метод checkFormValidation() перевіряє чи відображається форма для вводу номеру замовлення та електронної пошти користувача. Якщо, поля для вводу

відображаються, то метод отримує необхідні дані для подальшої перевірки. Якщо була виявлена помилка валідації даних, то відбувається перевірка того чи відображається повідомлення про помилку. Якщо, користувач ввів валідні дані, але такого замовлення не існує, метод перевіряє чи відображається повідомлення про те, що замовлення не знайдене.

Метод checkOrderDetails() перевіряє чи правильні дані відображаються в знайденому замовленні.

Метод fillOrderIdInput() приймає два параметри, isIdReal та isValid, в залежності в значень параметрів вводить номер замовлення в поле для вводу номера замовлення.

Метод fillEmailInput() приймає два параметри, isEmailReal та isValid, в залежносі від значень параметрів, вводить електронну пошту в поле для вводу електронної пошти.

Клас *newUserCheckoutPage* описує сторінку завершення створення замовлення для користувача який вирішив зареєструватися в системі.

В класі розроблені наступні методи:

– checkInputValidation();

– методи типу fill().

Розглянемо методи класу newUserCheckoutPage детальніше.

Метод checkInputValidation() знаходить всі поля в які користувач може вводити дані. Далі метод перевіряє чи знайдене поле обов'язкове чи ні. Якщо поле обов'язкове, то метод зберігає небхідні дані для подальшої перевірки. Якщо в обов'язковому полі була виявлена помилка валідації, то метод перевіряє чи відображається повідомлення про помилку.

Методи типу fill() використовуються для внесення даних до полів вводу.

Клас *guestCheckoutPage* описує сторінку завершення створення замовлення для користувача який не планує реєструватися в системі.

В класі розроблені наступні методи:

– checkOrderSummary();

– checkInputsValidation();

– fill().

Розглянемо методи класу guestCheckoutPage детальніше.

Метод checkOrderSummary() отримує дані про товари які відображаються на сторінці і в кошику, та порівнює їх. Товари повинні співпадати.

Метод checkInputsValidation() знаходить всі поля в які користувач може вводити дані. Далі метод перевіряє чи знайдене поле обов'язкове чи ні. Якщо поле обов'язкове, то метод зберігає небхідні дані для подальшої перевірки. Якщо в обов'язковому полі була виявлена помилка валідації, то метод перевіряє чи відображається повідомлення про помилку.

Методи типу fill() використовуються для внесення даних до полів вводу.

Клас *forgotPasswordPage* описує сторінку відновлення паролю.

В класі розроблені наступні методи:

– checkInputValidation;

– методи типу fill().

Розглянемо методи класу forgotPasswordPage детальніше.

Метод checkInputValidation знаходить всі поля в які користувач може вводити дані. Потім метод зберігає небхідні дані про знайдені поля для подальшої перевірки. Якщо в полі була виявлена помилка валідації, то метод перевіряє чи відображається повідомлення про помилку.

Методи типу fill() використовуються для внесення даних до полів вводу.

3.3 Розробка тест-кейсів

Тест-кейс – це тестовий артефакт, суть якого полягає у виконанні деякої

кількості дій та/або умов, необхідних для перевірки певної функціональності програмної системи, що розробляється [6].

Позитивний тест-кейс №1: додати товар у кошик та потім видалити товар. В таблиці 3.1 розроблено позитивний тест-кейс.

Таблиця 3.1 – Позитивний тест-кейс № 1

Кроки	Очікуваний результат
Відкрити сайт https://automationteststore.com/	Відкривається головна сторінка
Вибрати товар, та додати його до кошику	Доданий товар має відобразитися у верхньому кошику
Натиснути на кнопку “View Cart”	Відкриється сторінка “Shopping Cart”
Натиснути на кнопку “Remove”	Доданий товар має зникнути

Програмна реалізація тест-кейсу № 1.

Для реалізації цього тест-кейсу треба звернутися до класу shoppingCartPage, та викликати методи visitPage(), checkCartTable(), removeItemFromCart(), checkCartTable(). Також необхідно викликати функцію AddProductToCart().

```
it('Positive scenario: Add one product to cart and remove it', function(){

    cy.AddProductToCart(0)

    shoppingCartPage
        .visitPage()
        .checkCartTable()
        .removeItemFromCart()
        .checkCartTable()
})
```

Рисунок 3.2 – Програмна реалізація тест-кейсу № 1

Негативний тест-кейс № 2: перевірити деталі замовлення використовуючи неіснуючий номер замовлення, та електронну пошту, яка використовувалась для створення замовлення. В таблиці 3.2 розроблено негативний тест-кейс.

Таблиця 3.2 – Негативний тест-кейс № 2

Кроки	Очікуваний результат
Відкрити сайт https://automationteststore.com/	Відкривається головна сторінка
Натиснути на кнопку “Check Your Order”	Відкриється сторінка “Order Details”
Ввести неіснуючий номер замовлення	Введений номер відображається на сторінці
Ввести електронну адресу яка використовувалась для створення замовлення	Введена електронна адреса відображається на сторінці
Натиснути на кнопку “Continue”	Відобразиться повідомлення що таке замовлення не знайдене

Програмна реалізація тест-кейсу № 2.

Для реалізації цього тест-кейсу треба звернутися до класу mainPage, та використати метод visitMainPage(). Також необхідно звернутися до класу orderDetailsPage, та викликати методи openOrderDetailsPage(), fillOrderIdInput(!isIdReal, !isIdValid), fillEmailInput(isEmailReal, isEmailValid), clickContinueButton(), checkFormValidation().

```
it('Negative scenairo: Enter unreal invalid order id, enter real valid email', function(){

    mainPage
        .visitMainPage()

    orderDetailsPage
        .openOrderDetailsPage()
        .fillOrderIdInput(!isIdReal, !isIdValid)
        .fillEmailInput(isEmailReal, isEmailValid)
        .clickContinueButton()
        .checkFormValidation()

})
```

Рисунок 3.3 – Програмна реалізація тест-кейсу № 2

Позитивний тест-кейс № 3: створити замовлення гостьовим користувачем. В таблиці 3.3 розроблено позитивний тест-кейс.

Таблиця 3.3 – Позитивний тест кейс № 3

Кроки	Очікуваний результат
Відкрити сайт https://automationteststore.com/	Відкривається головна сторінка
Вибрати товар, та додати його до кошику	Доданий товар має відобразитися у верхньому кошику
Натиснути на кнопку “View Cart”	Відкриється сторінка “ShoppingCart”
Натиснути на кнопку “Checkout”	Відкриється сторінка “Account Login”
Вибрати “Guest Checkout” та натиснути на кнопку “Continue”	Відкриється сторінка для заповнення даних
Заповнити обов'язкові поля	Введена інформація відображається на сторінці
Натиснути на кнопку “Continue”	Відкриється сторінка “Checkout Confirmation”

Продовження таблиці 3.3

Кроки	Очікуваний результат
Натиснути на кнопку “Confirm Order”	Відкриється сторінка “You order has been processed

Програмна реалізація тест-кейсу № 3.

Для реалізації цього тест-кейсу треба звернутися до класу shoppingCartPage, та викликати методи visitPage(), clickCheckoutButton(). Також необхідно звернутися до класу guestCheckoutPage, та методам selectGuestCheckoutRadioButton(), clickContinueButton(), checkOrderSummary(), checkInputsValidation(), fillMethods(), clickConfirmOrderButton().

```
    it('Positive scenario', function(){

        cy.AddProductToCart(0)

        shoppingCartPage
            .visitPage()
            .clickCheckoutButton()

        guestCheckoutPage
            .selectGuestCheckoutRadioButton()
            .clickContinueButton()
            .checkOrderSummary()
            .clickContinueButton()
            .checkInputsValidation()
            .fillFirstNameInput(guestCheckoutPageData.firstName)
            .fillLastNameInput(guestCheckoutPageData.lastName)
            .fillEmailInput(isEmailValid)
            .fillAddress1Input(guestCheckoutPageData.address1)
            .fillCityInput(guestCheckoutPageData.city)
            .fillRegionSelect(3)
            .fillZipCodeInput(guestCheckoutPageData.zipCode)
            .clickContinueButton()
            .clickConfirmOrderButton()
    })
})
```

Рисунок 3.4 – Програмна реалізація тест-кейсу № 3

3.4 Запуск розроблених автотестів в Jenkins

Jenkins – це інструмент автоматизації з відкритим вихідним кодом, написаний на Java з плагінами, створеними для постійної інтеграції. Jenkins використовується для створення та постійного тестування проектів програмного забезпечення, що полегшує розробникам інтеграцію змін у проект, а користувачам – отримати нову збірку. Це також дозволяє безперервно постачати програмне забезпечення завдяки інтеграції з великою кількістю технологій тестування та розгортання [7].

З Jenkins організації можуть прискорити процес розробки програмного забезпечення за рахунок автоматизації. Дженкінс інтегрує всі види процесів життєвого циклу розробки, включаючи збірку, документування, тестування, упаковку, етап розгортання, статичний аналіз і багато іншого [7].

Переваги Jenkins:

– це інструмент із відкритим вихідним кодом, який має велику підтримку спільноти;

– його легко встановити;

– він має понад 1000 плагінів, щоб полегшити роботу (якщо плагін не існує, ви можете стровирити плагін сомостійно та поділитися ним із спільнотою);

– jenkins безкоштовний;

– він створений на основі Java і, отже, переноситься на всі основні платформи.

Для запуску автотесті в Jenkins треба строрити новой проєкт або job. Jenkins job – це послідовний набір завдань, які визначає користувач. Наприклад, завдання може отримати вихідний код із системи керування версіями, скомпілювати код, запустити модульні тести тощо [8].

На рисунку 3.5 наведено приклад створення нового проєкту в Jenkins. Для

цього необхідно перейти на сторінку Dashboard та натиснути на кнопку New Item. Далі необхідно назвати проєкт та вибрати його тип. Після цього необхідно виконати додаткову конфігурацію, приєднати Git Hub репозиторій та визначити скрипти запуску білда.

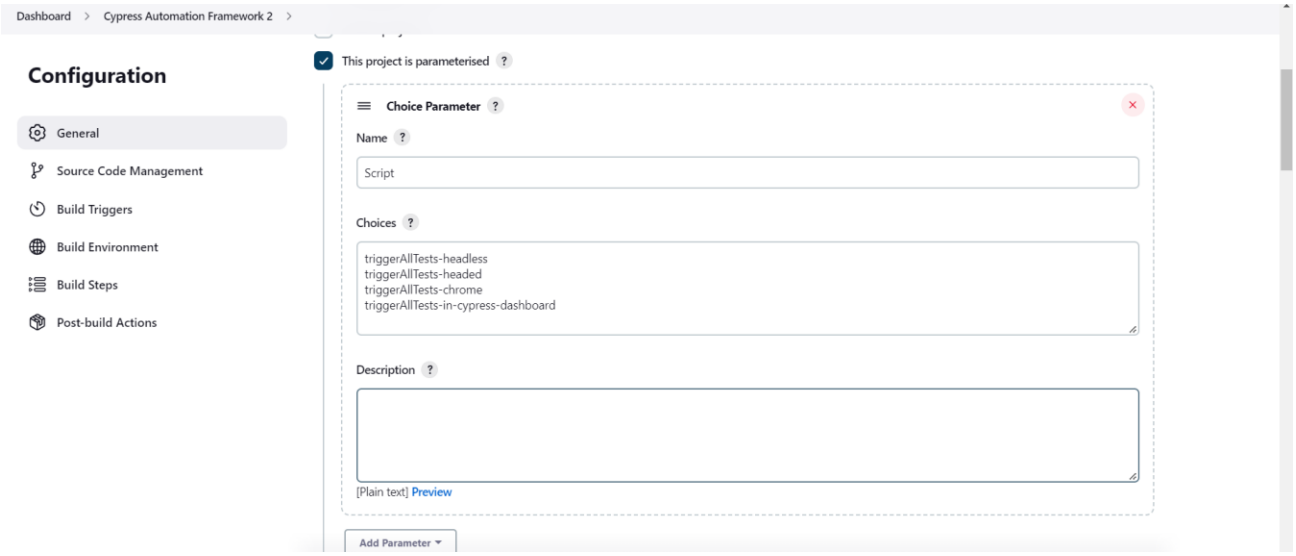

Рисунок 3.5 – Конфігурія нового проєкту

Коли проєкт створений та відукофігурований можна спробувати його запустити. На рисунку 3.6 наведено приклад результату виконання автотестів.

Jenkins дуже зручний тим що можна визначати коли саме будуть запускатися автотести.

Можуть бути використані наступні варіанти:

– запустити завдання вручну;

– періодично створюйте його, використовуючи креоновий синтаксис;

– почніть виконання роботи після завершення іншого проекту;

– опитуйте контроль версії вихідного коду.

В моєму випадку використовується запуск авторесті вручну.

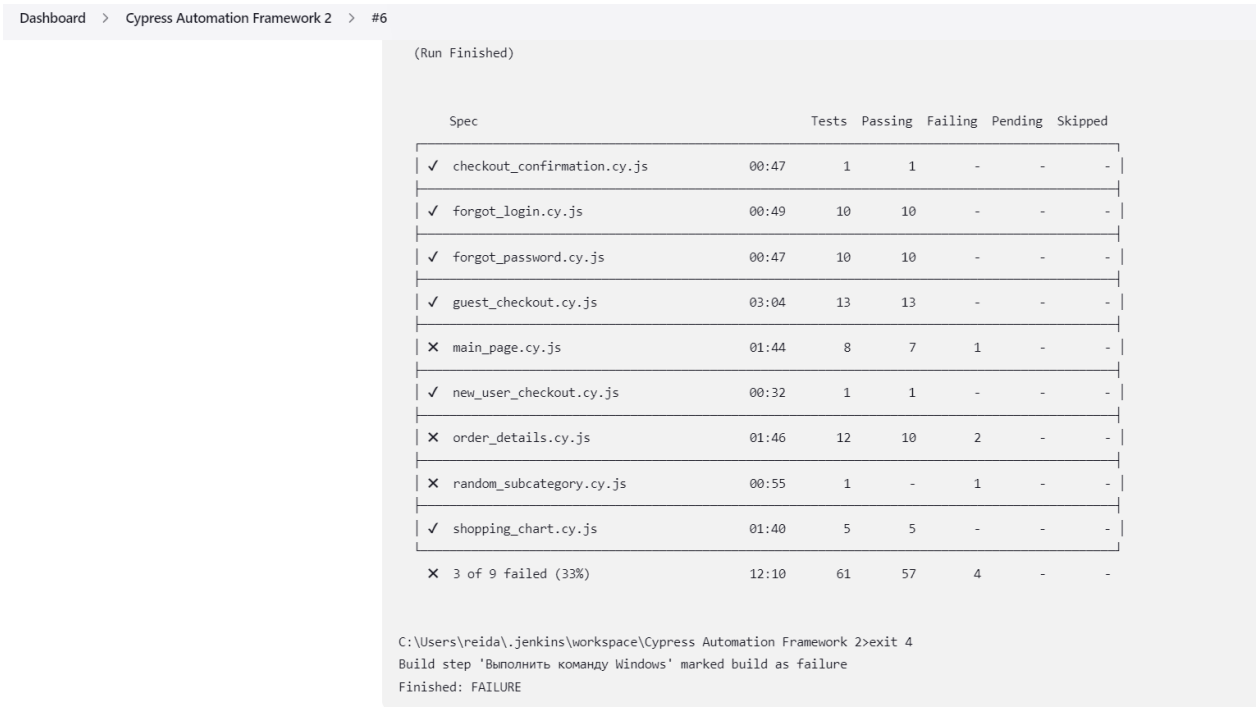

Рисунок 3.6 – Результат виконання автотестів в Jenkins

3.5 Візуалізація результатів в Cypress Dashboard

Інформаційна панель Cypress надає доступ до всіх створених тестів, зазвичай це відбувається, коли запускаються тести Cypress від свого постачальника постійної інтеграції. Інформаційна панель надасть вам уявлення про ефективність вашого тесту [9].

Інформаційна панель надає інтерфейс, де можливо:

– налаштувати проект, який ви хочете записати, на інформаційній панелі;

– налаштувати контроль доступу до проекту Cypress;

– передавати право власності на проекти.

На рисунку 3.7 зображен результат прогону автотестів. Всього було розроблено 61 тест. Опираючись на статистику можна побачити що 56 із них були вдалими, а в 5 були виявлені помилки.

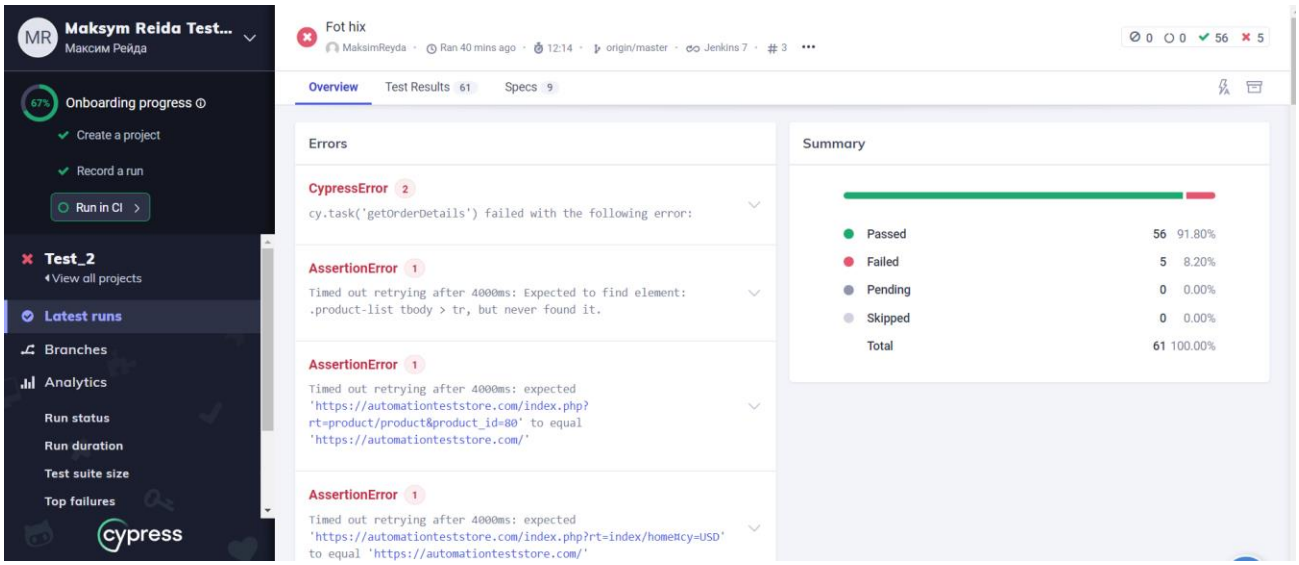

Рисунок 3.7 – Результат прогону автотестів

Інформаційна панель надає розгорнуту інформацію про тести в яких виникла помилка. На рисунку 3.8 зображено прикдад тесту в якому була виявлена помилка.

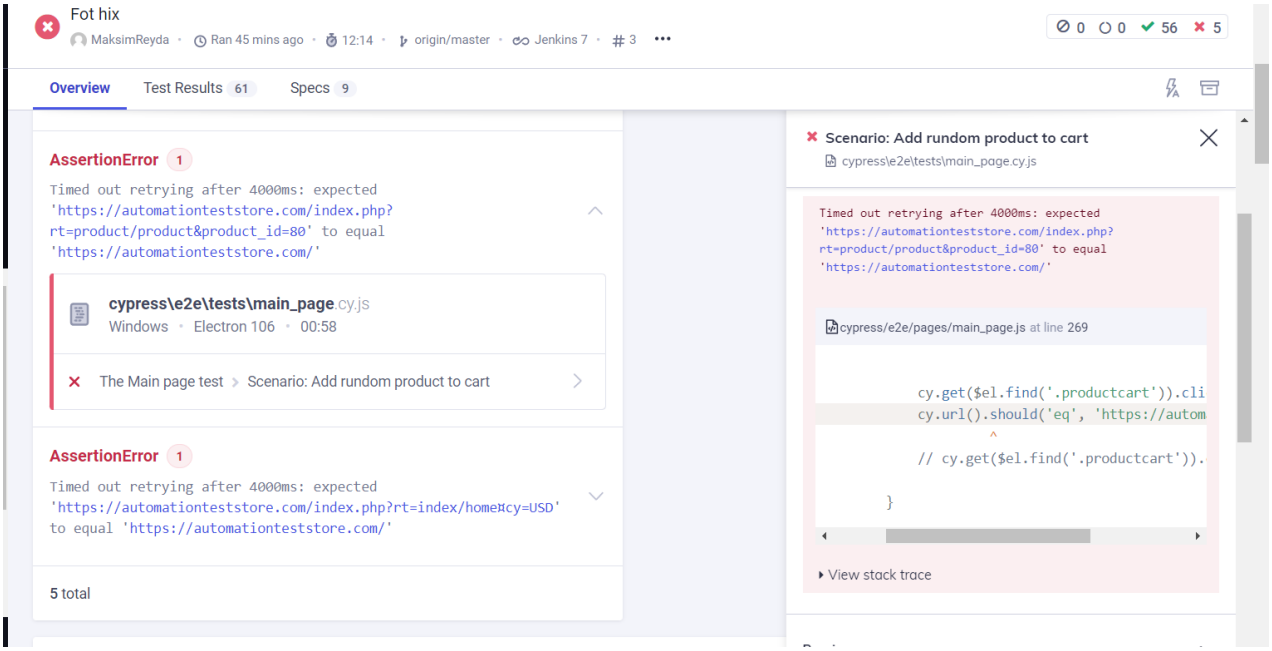

Рисунок 3.8 – Приклад тесту в якому була виявлена помилка

Система повідомляє в якому саме файлі та тесті помилка. В данному випадку іде мова про тест головної сторінки, а саме додавання випадкового товару до кошику. Помилка виникла через те що, після натискання на кнопку “Add to cart” сайт відкрив сторінку делай товару, замість того щоб залишитись на головній сторінці сайту.

Також інформаційна панель зберігає знімки екрану де саме виникла помилка. При додатковій конфігурації, система буде зберігати відео.

3.6 Висновки до третього розділу

В третьому розділі було описано сайт який використовується для розробки автотестів. Описані розроблені класи та їх методи. До методів додан опис дій які вони виконують. Наведено приклад трьох розроблених тест кейсів, та відображена їх програмна реалізація. Створено новий проєкт в Jenkins, та поєднано його з репозиторієм у Git Hub. Виконано прогін автотестів, та відображено результат. Також використано Cypress Dashboard для відображення результатів тестів.

ВИСНОВКИ

В ході виконання дипломної роботи було проаналізовано переваги та недоліки різних підходів до тестування. Охарактеризовані аспекти підходів та методів тестування програмного забезпечення. Розкриті теоретичні поняття, що стосуються теми тестування веб-додатків, і на підставі цих даних була розроблена система тестування.

Було розглянуто найпопулярніші на сьогоднішній день інструменти тестування вебдодатків. Аналізуючи зібрані дані, було виявлено слабкі і сильні сторони цих інстументів. Між деякими із них була виконана порівняльна характеристика. Також були розреблені діаграми які відображають використання сайту користувачем.

Було налаштовано систесу безперервної інтенрації. Також були розроблені тест кейси, а потім воконана їх програмна реалізація. Відображені результати прогону тестів із використанням Cypress Dashboard.

До основних плюсів автоматизації можна віднести можливість запускати автотести вбудь – який час доби на різних пристроях, що дозволяє проводити автоматичне тестування паралельно зручним.

Отже, на основі проведеного дослідження було виявлено, що автоматизоване тестування програмного забезпечення за допомогою необхідних інструментів вимагає меншого часу виконання автотестів. Але по вартості відносно більше автоматизоване, аніж ручне.

ПЕРЕЛІК ПОСИЛАНЬ

1. Рівні тестування. URL: https://www.javatpoint.com/levels-of-testing (дата звернення 12.07.2022).

2. Методи тестування. URL: https://www.tutorialspoint.com/software_testing/software_testing_methods.htm (дата звернення 15.07.2022).

3. Найбільш полулярні інструменти для автоматизіції тестування. URL: https://disbug.io/en/blog/automated-web-testing-tools (дата звернення 20.08.2022).

4. Що таке Cypress. URL: https://www.perfecto.io/blog/cypress-testing (дата звернення 21.08.2022).

5. Cypress та Selenium, який фреймворк кращий для тестування. URL: Cypress vs. Selenium: What's the Better Testing Framework? (testim.io) (дата звернення 25.08.2022).

6. Поняття тест кейсу. URL: https://qalight.ua/baza-znaniy/test-case-2/ (дата звернення 02.09.2022).

7. Що таке Jenkins. URL: https://www.edureka.co/blog/what-is-jenkins/ (дата звернення 03.09.2022).

8. Що таке job в Jenkins. URL: https://anto.online/code/jenkins-an-introduction-to-jobs-and-projects/#:~:text=A%20Jenkins%20job%20is%20a,is%20synonymous%20with%20%E2%80%9Cproject%E2%80%9D (дата звернення 10.09.2022).

9. Що таке Cypress Dashboard. URL: https://www.w3resource.com/cypress/dashboard-introduction.php#:~:text=The%20Cypress%20Dashboard%2C%20gives%20you,the%20performance%20of%20your%20test (дата звернення 15.09.2022).