ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

ПОЯСНЮВАЛЬНА ЗАПИСКА
до бакалаврської роботи
на ступінь вищої освіти бакалавр

на тему: «РОЗРОБКА ЗАСОБІВ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ WEB-САЙТУ КОМПАНІЇ REZET З ВИКОРИСТАННЯМ ТЕХНОЛОГІЙ JS, CYPRESS WEBDRIVER»

Виконав: студент 4 курсу, групи ПД–42
спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності/спеціалізації)

Волошин В.В.
(прізвище та ініціали)

Керівник Негоденко О.В.
(прізвище та ініціали)

Рецензент

(прізвище та ініціали)

Київ – 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти - «Бакалавр»
Спеціальність підготовки – 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Негоденко О.В.
"____"_____ 2023 року

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТА

Волошину Віталію Віталійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи: «Розробка засобів автоматизованого тестування web-сайту компанії Rezet з використанням технологій JS, Cypress webdriver»

Керівник роботи: Негоденко Олена Василівна, к.т.н, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом вищого навчального закладу від «24» лютого 2023 року №26.

2. Строк подання студентом роботи «1» червня 2023 року

3. Вхідні дані до роботи:

3.1 Науково-технічна література з питань, пов'язаних з програмним забезпеченням щодо розробки .
3.2 Існуючі інструменти автоматизації тестування ПЗ.
3.3 Науково-технічна література;
3.4 Положення побудови систем автоматизації тестування ПЗ;

4. Зміст розрахунково-пояснювальної записки.

4.1 Аналіз актуальності та проблематики розроблюваного навчального модулю.
4.2 Аналіз та вибір інструментів для реалізації навчального модулю.
4.3 Проектування навчального модулю.
4.4 Висновки.

5. Перелік демонстраційного матеріалу
5.1 Титульний слайд.
5.2 Мета, об’єкт, предмет, наукова новизна дослідження.
5.3 Актуальність.
5.4 Аналіз аналогів.
5.5 Технічні завдання.
5.4 Програмні засоби та інструменти реалізації.
5.6. Розробка архітектури.
5.7 Реалізація програми.
5.8 Висновки.
5.9 Апробація результатів дослідження.
5.10 Кінцевий слайд.
6. Дата видачі завдання «25» лютого 2023р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів	
роботи	Примітка		
1	Підбір науково-технічної літератури	25.02.2023	Виконано
2	Дослідження аналогів та актуальності додатку	13.03.2023	Виконано
3	Аналіз та вибір інструментів для розробки додатку	25.03.2023	Виконано
4	Проектування та реалізація	08.04.2023	Виконано
5	Вступ, висновки, реферат	02.05.2023	Виконано
6	Розробка обов‘язкових демонстраційних матеріалів	10.05.2023	Виконано
7	Попередній захист роботи	23.05.2023	Виконано

Студент Волошин В.В.
(підпис) (прізвище та ініціали)

Керівник роботи Негоденко О.В.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи с.46, табл.1, рис.16, джерел 20.

Ключові слова: автоматизоване тестування, Cypress, JavaScript, Rezet.

Об’єкт дослідження – процес автоматизації тестування web-сайту.

Предмет дослідження – засоби та технології для автоматизованого тестування web-сайту.

Мета роботи – спрощення процесу автоматизації тестування web-сайту компанії Rezet за рахунок використанням технологій Cypress та JavaScript.

Методи дослідження – методи передачі, зберігання та обробки інформації, уніфікований процес створення програмного забезпечення, емпіричні методи.

Для реалізації поставленої мети потрібно вирішити наступні завдання:

1. Проаналізувати архітектуру існуючих інструментів автоматизації тестування

2. Порівняти існуючі інструменти автоматизованого тестування для виявлення їх переваг та недоліків

3. Розробити програмне забезпечення, засобів автоматизованого тестування сайту компанії Rezet за допомогою Cypress

4. Провести тестування програмного забезпечення, щоб переконатися в його працездатності та відповідності вимогам

Практична значення результатів: Результати цього дипломного проекту матимуть практичне значення для індустрії програмного забезпечення, оскільки вони дозволять створити автоматизоване тестування , що забеспечить якість продукту та ефективний процес розробки. Використання такого засобу дозволить економити час та зусилля під час тестування, а також забезпечить точність перевірок і відповідність функціональним вимогам продукту.

Галузь використання – галузь розробки програмного забезпечення.

ЗМІСТ

ВСТУП

В сучасних умовах, де швидкість розвитку програмного забезпечення щоразу зростає, автоматизоване тестування є ключовим елементом процесу розробки. Програмна індустрія швидко змінюється, і вимагає швидкого та ефективного тестування, щоб забезпечити якість продукту та конкурентоспроможність на ринку.

Автоматизоване тестування дозволяє розробникам швидко виявляти та виправляти помилки, що забезпечує більш ефективний процес розробки. При цьому, воно забезпечує точність тестування, оскільки тести запускаються без участі людини, і тим самим усувається ризик людської помилки. Крім того, засоби автоматизованого тестування дозволяють автоматизувати процес тестування та зменшити час на тестування.

Тестування на різних платформах та з різних пристроїв є важливим елементом процесу розробки, оскільки воно дозволяє забезпечити коректну роботу додатку на різних пристроях та забезпечити задекларовану функціональність.

Зважаючи на усе вище сказане, розробка цього дипломного проекту буде нести практичну користь, оскільки його успішне виконання дозволить створити високоякісний продукт, який буде відповідати вимогам сучасного ринку програмного забезпечення.

Об’єкт дослідження – процес автоматизації тестування web-сайту.

Предмет дослідження – засоби та технології для автоматизованого тестування web-сайту.

Мета роботи – спрощення процесу автоматизації тестування web-сайту компанії Rezet за рахунок використанням технологій Cypress та JavaScript.

Методи дослідження – методи структурного аналізу і проектування, методи розробки програмного забезпечення, методи тестування програмного забезпечення, методи верифікації програмного забезпечення.

У цій дипломній роботі будуть розглянуті наступні завдання:

1. Проаналізувати архітектуру існуючих інструментів автоматизації тестування

2. Порівняти існуючі інструменти автоматизованого тестування для виявлення їх переваг та недоліків

3. Розробити програмне забезпечення, засобів автоматизованого тестування сайту компанії Rezet за допомогою Cypress

4. Провести тестування програмного забезпечення, щоб переконатися в його працездатності та відповідності вимогам

Результатом роботи буде засіб автоматизованого тестування сайту компанії Rezet, що дозволить забезпечити якість продукції та зменшити час, необхідний для тестування сайту.

Практичне значення результатів: Результати цього дипломного проекту матимуть практичне значення для індустрії програмного забезпечення, оскільки вони дозволять створити автоматизоване тестування , що забеспечить якість продукту та ефективний процес розробки. Використання такого засобу дозволить економити час та зусилля під час тестування, а також забезпечить точність перевірок і відповідність функціональним вимогам продукту.

Галузь використання - галузь розробки програмного забезпечення.

1. ОГЛЯД АРХІТЕКТУРИ ІСНУЮЧИХ ІНСТРУМЕНТІВ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ

1. 1 Переваги засобів автоматизації тестування над ручнимтестуванням

На початковій стадії розвитку програмної індустрії тестування здійснювалось виключно вручну. Однаковий характер тестування, однотипність завдань та тривалість процесу зробили очевидним те, що ринок потребує інструментів, які здатні виконувати тести від імені спеціаліста.

Проблеми регресії є найбільш болючими. Ми, як люди, не можемо робити те ж саме з однаковою енергією, швидкістю та точністю кожен день. Це можливо лише для машин. Тому для цього потрібна автоматизація, яка дозволяє повторювати ті самі дії з однаковою швидкістю, точністю та ефективністю.

Автоматизація тестування допомагає фахівцям зосередитись на тестуванні нових функціональних можливостей, одночасно покриваючи регресивне тестування системи. Згідно з дослідженнями ISTQB, автоматизоване тестування скорочує час виконання тестів до 90%.

Сценарій заповнює всі поля та показує результат запуску із скріншотами. Якщо тест не вдався, він може точно визначити місце, де відбулася помилка, і таким чином допомогти вам легко її відтворити.

Автоматизація є економічно ефективним методом регресійного тестування. Спочатку витрати на автоматизацію справді вищі: вони включають вартість інструменту, вартість персоналу для автоматизації тестування та їх навчання. Але коли сценарії готові, їх можна виконувати багаторазово з однаковою точністю і швидкістю. Це заощаджує багато робочого часу фахівців. Тому вартість впровадження автоматизації поступово зменшується, і в кінцевому рахунку це стає економічно ефективним методом регресійного тестування.

Завдяки автоматизації тестування тестувальники можуть автоматизувати повторювані завдання та сконцентруватись на інших тестових завданнях, таких як вибір правильних тестових сценаріїв для тестового запуску та тестування нових

функцій. Автоматизоване тестування дозволяє зменшити час, необхідний на тестування повторюваних сценаріїв, але такий підхід є недосконалим, оскільки він передбачає використання скриптів, що потребує спеціальних знань та навичок програмування. Крім того, використання автоматизованого тестування на основі сценаріїв може вимагати великих початкових інвестицій у налаштування засобів тестування та призначення ресурсів для автоматизації, а також стає високооб'ємним завданням підтримувати автоматизовані тест-кейси. Для вирішення цих проблем на ринку існують засоби автоматизованого тестування, які не вимагають скриптів, такі як інструменти «запису та відтворення», які стають все більш популярними серед компаній.

Багато підприємств зосередили свою увагу на безперервному тестуванні, використовуючи автоматизоване тестування, щоб уникнути критичних бізнес– проблем. Крім того, для досягнення конкурентних переваг їм потрібно створювати свій продукт / додаток у постійному стані готовності до випуску.

При розробці додатків або в життєвому циклі розробки програмного забезпечення існує безліч сценаріїв та тригерних точок, які вимагають застосування автоматизації тестів для наскрізного тестування.

Наприклад, рішення робототехніки та ботів надзвичайно важливо протестувати якісно, щоб переконатися, що продуктивність відповідає очікуванням. Або ж це може призвести до руйнівного провалу у використанні цієї технології.

Автоматизація тестів дозволяє повторно використовувати сценарії тестів та максимально використовувати покриття тесту, щоб пришвидшити процес тестування з точністю та надійністю. Це зменшує зусилля людини і одночасно гарантує, що додаток працюватиме належним чином, як очікувалося на стадії передачі кінцевому користувачу.

Доступні різні інструменти автоматичного запису та відтворення тестування, завдяки яким полегшується створення тесту. Це розширення браузера / плагіни, що використовується для запису заздалегідь визначених дій, які виконують тестувальники під час взаємодії з веб– додатком. Тоді як функція відтворення

автоматично генерує попередньо записані дії, щоб врешті порівняти результати з очікуваним результатом.

Цей інструмент реєструє кожну дію тестера, наприклад, наведення миші, знімки екрана та натискання клавіш під час виконання тест– кейса вручну. Коли тест записаний, його можливо запускати заново, коли потрібно зробити регресійне тестування системи. Простіше кажучи, цей інструмент дозволяє виконувати одні й ті самі сценарії кілька разів.

Ось причини, чому слід розглянути можливість записування та відтворення для автоматичного тестування:

Інструменти запису і відтворення – це безпроблемний спосіб автоматизувати процеси ручного тестування в найпростішій формі. Вони не вимагають знань програмування. Ви можете виконати ручне тестування без необхідності писати довгі сценарії, і функція запису буде реєструвати ці дії.

Цей інструмент дуже ефективний для тестувальників, які перебувають на етапі навчання; без необхідності боротися зі складними фреймворками та командами

Автоматизація запису та відтворення дозволяє створювати, редагувати та імпортувати записані сценарії, а також відстежувати процеси тестування. Це дає змогу легко перезаписувати або відредаговувати частину записаних сценаріїв, якщо помилка виявиться під час процесу запису.

Користувачі інструменту запису та відтворення можуть перевіряти функціональність в різних браузерах, використовуючи різні конфігурації.

Інструменти запису та відтворення корисні в початковому тестуванні програми, але роль спеціальної команди з контролю якості розмивається. Кожен може виконати початкове тестування за допомогою цих інструментів, будь то розробник або менеджер. Це зменшує навантаження на спеціальну команду з тестування та дозволяє зосередитись на інших важливих тестових заходах. Інструменти запису та відтворення в тестовій автоматизації мають вирішальне значення для розробки додатків та циклу розробки програмного забезпечення. Це прискорює процес тестування програмного забезпечення, а також робить його

більш ефективним та орієнтованим на результат.

1.2 Розгляд наявних засобів, які використовуються для автоматизації тестування програмного забезпечення

Інструменти запису та відтворення

Інструменти автоматизації тестування, які використовують запис і відтворення тестових сценаріїв, є корисними для стабільних сценаріїв, що не зазнають змін тривалий час. Але якщо програмне забезпечення нестабільне та піддається частим змінам, то запуск тестів може бути неуспішним.

Головним недоліком використання таких інструментів полягає в тому, що вони можуть записувати та відтворювати тільки тестові сценарії, які були заздалегідь визначені. Це обмежує можливість налаштування записаних тестів. Деякі засоби, наприклад Selenium IDE та Katalon IDE, дозволяють конвертувати записані тестові сценарії у скрипти на обраних мовах програмування, але для редагування таких тестів необхідні відповідні програмувальні навички.

Інструменти автоматизації тестування, такі як Testsigma, використовують NLP (Natural Language Processing) для створення тестових сценаріїв, що робить користування дуже легким і доступним. Оскільки автоматизовані тести написані зазвичай простою природною мовою, користувачеві необхідно знати правильну граматику, яка використовується для розробки тестів, і використовувати її для автоматизації як простих, так і складних сценаріїв тестування, залежно від конкретної ситуації. Це дозволяє значно знизити поріг входження для користувачів і спрощує процес створення тестових сценаріїв.

Selenium - це набір інструментів для автоматизації тестування веб-додатків. Він дозволяє розробникам тестів писати скрипти на різних мовах програмування, таких як Java, Python, Ruby тощо, щоб автоматизувати тестування веб-додатків.

Selenium дозволяє емулювати реальну поведінку користувача, навіть у випадку, коли веб-додаток містить складні динамічні елементи, такі як AJAX, JavaScript та інші. Це може бути корисно для тестування сайтів з великою кількістю

взаємодій, таких як електронні магазини, банківські системи тощо.За допомогою Selenium можна виконувати різні види тестів, такі як функціональні, регресійні, UI та інші. Крім того, він дозволяє автоматизувати тестування на різних браузерах, таких як Google Chrome, Mozilla Firefox, Microsoft Edge тощо. Selenium є одним з найбільш популярних інструментів для автоматизації тестування веб-додатків, і він постійно оновлюється та розвивається.

TestNG (Test Next Generation) є фреймворком для автоматизованого тестування Java-додатків. Він надає більш широкий та гнучкий функціонал, ніж стандартна бібліотека JUnit для автоматизованого тестування. TestNG дозволяє створювати та запускати тести, які можуть бути груповані та параметризовані. Він також підтримує функції, такі як паралельне виконання тестів, тестування залежностей та звітність про результати тестування. TestNG можна використовувати для тестування як на рівні модуля, так і на рівні системи. Він може бути інтегрований з такими інструментами, як Maven та Jenkins, для автоматичного запуску тестів при збірці та розгортанні додатка.

Візуальне тестування за допомогою блок– схем

За допомогою Codefuse користувачі можуть створювати тести у вигляді блок-схем, де кожен блок відповідає дії на веб-сайті. Такий підхід не вимагає створення тестових сценаріїв, що робить процес тестування більш простим та ефективним.

Запис тестів як сценаріїв

Цей метод тестування використовується в Selenium IDE та Katalon IDE і полягає у записі усіх користувацьких дій у вигляді скрипту. Кожен записаний тестовий сценарій містить команди та деталі селектора. Тестові сценарії можна експортувати у вибрану мову та, при необхідності, розширити цією мовою.

AI– боти для автоматизації тестів

Ці інструменти дозволяють уникнути потреби в складних сценаріях для автоматизації тестування. На ринку існує декілька інструментів, які навчають ботів відтворювати дії з вхідними даними за зразком. Процес навчання безперервний і не потребує складних програмних навичок.

1.3 Розробка автоматизованого тестування для сайту за допомогою мови JS

Розробка автоматизованих тестів за допомогою мови JavaScript (JS) є досить популярним підходом у тестуванні програмного забезпечення. JS має широко розповсюджений складній стек технологій, що дозволяє розробникам створювати високоякісні та ефективні автоматизовані тести.

Для розробки автоматизованих тестів за допомогою JS можна використовувати різні фреймворки та бібліотеки, такі як Jest, Mocha, Jasmine, Protractor тощо. Кожен з них має свої переваги та недоліки, але загалом всі вони дозволяють розробляти тести в зручний та ефективний спосіб.

Для автоматизованого тестування веб-додатків за допомогою JS можна використовувати фреймворки, такі як Cypress та WebDriverIO. Cypress дозволяє розробляти тести на високому рівні абстракції, що забезпечує більш просту та зрозумілу структуру тестів. WebDriverIO, з іншого боку, дозволяє використовувати багато різноманітних браузерів для тестування, що робить його більш універсальним фреймворком.

Для створення автоматизованих тестів за допомогою JS також можна використовувати бібліотеки для роботи з DOM (Document Object Model), такі як jQuery або cheerio. Вони дозволяють звертатись до елементів сторінки та здійснювати з ними різноманітні операції, такі як клік, введення тексту, отримання значень тощо.

Загалом, розробка автоматизованих тестів за допомогою мови JavaScript є досить простою та ефективною, завдяки широкої підтримці різних фреймворків та бібліотек. Вона дозволяє розробникам створювати в исокоякісні та стійкі до змін тести для своїх веб-додатків. Крім того, мова JS є досить простою та зрозумілою для багатьох розробників, що зробило її досить популярною для автоматизованого тестування.

Однією з переваг розробки тестів на мові JS є їхній швидкий запуск та виконання, що дозволяє швидко виявляти помилки та проблеми в програмному

забезпеченні. Крім того, багато фреймворків та бібліотек для розробки тестів мають вбудовані засоби для створення звітів та аналізу результатів тестування, що робить процес тестування ще більш ефективним та зручним.

Окрім того, розробка тестів за допомогою мови JS дозволяє розробникам створювати тести на різних рівнях (unit тести, integration тести, end-to-end тести) та використовувати різні підходи до тестування (наприклад, тестування на реальних пристроях або за допомогою емуляторів та симуляторів).

В цілому, розробка автоматизованих тестів за допомогою мови JS є досить простою та ефективною, а також дозволяє розробникам створювати високоякісні та стійкі до змін тести для своїх веб-додатків. При цьому, важливо враховувати особливості конкретного проекту та вибрати найбільш підходящі фреймворки та бібліотеки для його автоматизованого тестування.

1.4 Огляд інструменту автоматизації тестування Cypress

Для розробки засобів автоматизованого тестування було вибрано інструмент Cypress

Cypress - це інструмент автоматизованого тестування програмного забезпечення (QA), який дозволяє створювати, запускати та проводити автоматизовані тести для веб-додатків. Cypress є відкритим програмним забезпеченням та працює на платформі Node.js.

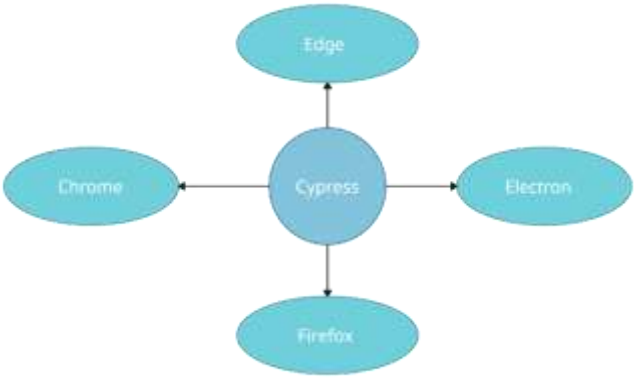


Рисунок.1.1. – Підтримувані драйвери браузерів Cypress.

Cypress надає наступні основні функції:

Автоматизоване тестування веб-додатків: Cypress дозволяє розробникам автоматизовано тестувати веб-додатки на різних пристроях та браузерах.

Відстеження помилок: Cypress дозволяє відстежувати та аналізувати помилки в реальному часі під час виконання тестів.

Легкість використання: Cypress надає інтуїтивно зрозумілий інтерфейс та просту структуру проекту, що дозволяє розробникам швидко створювати та запускати тести.

Інтерактивне тестування: Cypress дозволяє розробникам взаємодіяти з веб-додатком під час виконання тестів, що дозволяє виявляти та виправляти помилки швидше.

Спрощена настройка: Cypress дозволяє легко налаштовувати тести та швидко змінювати їх налаштування.

Різноманітність команд: Cypress надає широкий спектр команд, які дозволяють розробникам контролювати поведінку тестів, створювати фільтри, додавати налаштування та багато іншого.

Інтеграція з іншими інструментами: Cypress дозволяє інтегрувати тести з іншими інструментами для забезпечення неперервного інтегрування та постійної доставки.

Ці функції дозволяють тестувальникам та розробникам швидко та ефективно тестувати веб-додатки на Cypress, забезпечуючи високу якість коду та зменшуючи ризики помилок в продукції.

Cypress має кілька унікальних характеристик, які відрізняють його від інших інструментів автоматизації тестування:

– Інтерактивність є однією з ключових особливостей Cypress, яка забезпечує зручну взаємодію з веб-додатком під час виконання тестів. Даний фреймворк дозволяє розробникам маніпулювати станом елементів на сторінці, взаємодіяти з формами, кнопками та іншими елементами веб-додатку, вводити дані та перевіряти їх. Це значно полегшує процес розробки тестів та дозволяє забезпечити більш глибоку перевірку функціональності веб-додатку.

Додатково, ця функціональність дозволяє знайти більше помилок, які можуть бути пропущені звичайними засобами автоматизації тестування. Розробник може здійснювати взаємодію з веб-додатком у реальному часі, що дозволяє значно збільшити ефективність процесу тестування.

Також, за допомогою цієї функціональності, розробник може більш точно визначити місце та причину помилок, що дозволяє забезпечити більш якісне тестування веб-додатку. У разі виникнення помилок, розробник може застосувати необхідні зміни в коді тестів або веб-додатку в режимі реального часу, що дозволяє значно зменшити час, необхідний для виявлення та виправлення помилок.

Відстеження помилок в режимі реального часу: Cypress дозволяє відстежувати та аналізувати помилки в реальному часі під час виконання тестів. Це дозволяє швидко виявляти та виправляти проблеми..

Cypress має дуже зручний та простий синтаксис, що дозволяє не тільки швидко писати тести, але й забезпечує зрозумілість коду для всього команди. Цей синтаксис базується на ланцюжках методів, що дозволяє легко налаштувати тести та забезпечує зрозумілість коду.

Наприклад, Cypress дозволяє використовувати методи, які називаються "chaining", що дозволяє ланцюжити методи разом та створювати більш читабельний та логічний код. Крім того, Cypress підтримує такі різноманітність методів, які дозволяють легко взаємодіяти з елементами сторінок та перевіряти їх стан, такі як .click (), .type (), .select (), .check () та інші.:

Рисунок.1.2. – Принцип взаємодії Cypress з браузером.

Локація веб–елементів – для того, щоб виконувати такі дії, як натискання, друк, drag&drop, нам спочатку потрібно визначити, над яким веб–елементом (кнопка, прапорець, спадне меню, текстове поле) нам потрібно виконати дію. Щоб

полегшити це, WebDriver реалізовує методи ідентифікації веб–елементів за допомогою різних атрибутів HTML – таких як ідентифікатор, ім'я, клас, CSS, ім'я тегу, XPath, текст посилання тощо.

Обробка динамічних веб–елементів – Бувають випадки, коли на сторінціє веб– елементи, які змінюються при кожному перезавантаженні сторінки. Оскільки атрибути HTML змінюються, виявлення цих елементів стає проблемою. Cypress пропонує різні способи обробки динамічних веб-елементів. Один з таких способів - це використання команди `cy.wait()`, яка дозволяє зупинити виконання тесту до тих пір, поки не з'явиться потрібний елемент. Наприклад, якщо потрібно перевірити наявність модального вікна, можна використовувати команду `cy.wait()` з параметром, який відповідає за час очікування.

Ще один спосіб - це використання команди `cy.get()`, яка дозволяє знайти елемент на сторінці та взаємодіяти з ним. Завдяки цьому можна отримати доступ до динамічно змінюваних елементів та взаємодіяти з ними.

Також Cypress пропонує використання функції `cy.intercept()`, яка дозволяє перехоплювати запити, що відправляються з браузера та повертати певні дані. Це може бути корисним для тестування, коли потрібно перевірити реакцію додатку на різні дані.

Узагалі, Cypress пропонує багато різноманітних способів для обробки динамічних веб-елементів та забезпечення якісного тестування веб-додатків.

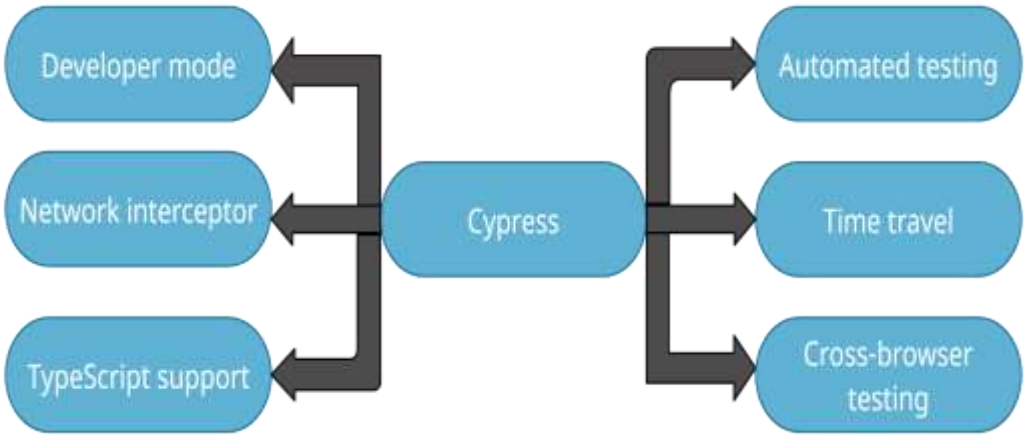


Рисунок.1.3. – Основні функції Cypress.

Cypress автоматично очищує стан тестового середовища перед кожним запуском тестів, що забезпечує стабільність виконання тестів. Це означає, що ви можете бути впевнені, що тести будуть запускатися в однакових умовах кожен раз, і результати будуть прогнозовані та надійні.

Іншими словами, не потрібно турбуватися про те, що попередні тести можуть вплинути на результати наступних тестів. Завдяки автоматичному очищенню стану, ви зможете ефективно тестувати свій продукт та впевнено знаходити та виправляти помилки.

Cypress - це інструмент, який складається з різноманітних компонентів, які діють як єдина система тестування. До складу цих компонентів входять різні інструменти, які необхідні для створення та запуску тестів. Оскільки ці компоненти взаємодіють між собою, Cypress не може бути розглянутий як окремий інструмент тестування, але як архітектурна складова, яка складається з різних компонентів. Крім того, Cypress автоматично очищує стан тестового середовища перед кожним запуском тестів, що забезпечує стабільність виконання тестів та прогнозовані та надійні результати.

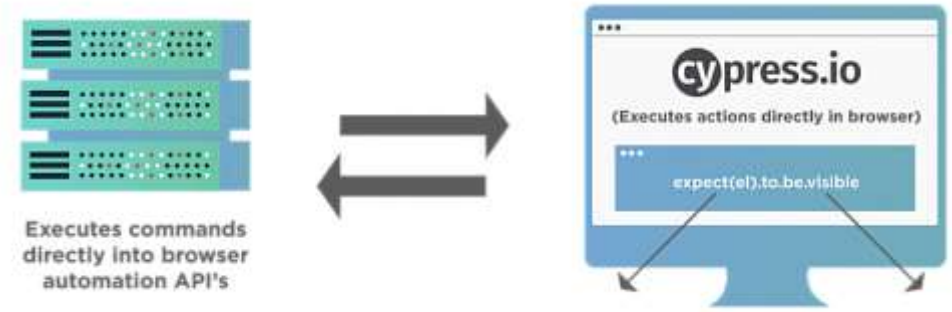


Рисунок.1.4. – Модель архітектури Cypress

Cypress Automation Protocol (CAP) - це протокол, розроблений спеціально для взаємодії між Cypress та браузером. Він є більш ефективним та потужним, ніж стандартний JSON Wire Protocol (JWP), який використовується багатьма іншими тестовими фреймворками, включаючи Selenium.

Однією з ключових переваг CAP є те, що він дозволяє Cypress взаємодіяти з

браузером без потреби запускати додатковий сервер, який працює з браузером (як це робиться в Selenium). Це дозволяє зменшити час на підготовку тестового середовища та робить процес тестування більш швидким та ефективним.

Крім того, CAP дозволяє Cypress працювати з динамічними елементами сторінок, такими як асинхронні запити та взаємодія з Ajax-елементами. Це робить тестування більш надійним та стійким до змін веб-додатку.

CAP також має вбудований таймер, який дозволяє автоматично чекати на відповідь від браузера. Це дозволяє запобігти помилкам, пов'язаним з часом відповіді веб-додатку, та робить тестування більш стабільним та менш вразливим до помилок. прогнозовані та надійні результати..

У Cypress є свій власний драйвер браузера, який називається Cypress Driver. Він побудований на базі Electron та розроблений спеціально для використання з Cypress.

Однією з ключових переваг Cypress Driver є те, що він повністю інтегрований з Cypress, що дозволяє забезпечити швидке та ефективне виконання тестів. Крім того, Cypress Driver автоматично перезавантажує сторінки при необхідності, що робить процес тестування більш стійким та менш вразливим до помилок.

За замовчуванням, Cypress підтримує Chromium-браузер, але також можна налаштувати використання інших браузерів, таких як Firefox або Edge.

Крім того, Cypress надає можливість взаємодіяти з зовнішніми драйверами браузера, якщо необхідно. Наприклад, ви можете використовувати Selenium для взаємодії з іншими браузерами, такими як Safari або Internet Explorer.

1.5 Архітектура сучасних Web сайтів

1.5.1 Використання NodeJS

Node.js є серверною платформою, яка використовує JavaScript Engine Google Chrome (V8 Engine) для створення швидких та масштабованих мережевих програм. Node.js використовує керовану подіями неблоковану модель вводу-виводу, що дозволяє розробникам ефективно створювати додатки, які можуть обробляти дані

у режимі реального часу на розподілених пристроях. Розроблений у 2009 році Райаном Далом, остання версія Node.js - v0.14.36.

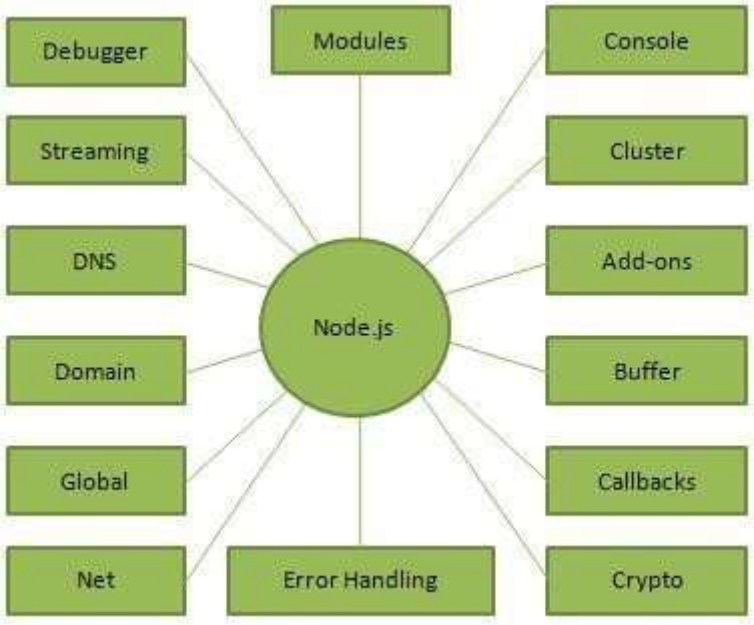


Рисунок 1.5. – Основні модулі платформи Node JS

Node.js є відкритим міжплатформеним середовищем виконання, призначеним для розробки серверних та мережевих додатків. Він дозволяє розробникам писати додатки на JavaScript і запускати їх на операційних системах Mac OS, Microsoft Windows та Linux. Node.js також надає багату бібліотеку різноманітних модулів JavaScript, яка спрощує розробку веб-додатків. Node.js має багато важливих функцій, включаючи асинхронність та керування подіями, що робить його ефективним для розробки швидких та масштабованих додатків. Бібліотека Node.js побудована на JavaScript Engine V8 від Google Chrome і може виконувати код на рівні з мовами програмування, такими як C# та Java..

Node.js - це мінімалістичне та ефективне середовище виконання, яке може бути використано для розробки серверних та мережевих додатків на різних операційних системах, таких як Mac OS, Microsoft Windows та Linux. В основі Node.js лежить бібліотека різноманітних модулів JavaScript, яка значно спрощує процес розробки веб-додатків.

Одним із головних переваг Node.js є асинхронність його API, тобто він не блокує виконання інших операцій, поки чекає на відповідь від попередньої операції. Це забезпечує швидкість роботи сервера, особливо в порівнянні з традиційними серверами, які створюють обмежені потоки для обробки запитів.

Node.js використовує однопоточну модель з циклічними подіями, що робить сервер дуже масштабованим. Він може обробляти набагато більше запитів, ніж традиційні сервери, такі як Apache HTTP Server. Крім того, програми Node.js ніколи не буферизують будь-які дані, а виводять їх шматками.

Node.js може бути використаний для розробки різноманітних програм, таких як програми з введенням/виведенням, потокового передавання даних, інтенсивні програми в режимі реального часу, додатки на основі JSON API та односторінкові програми. Однак, не рекомендується використовувати Node.js для побудови додатків, які вимагають інтенсивне навантаження на процесор, такі як обробка фото та відео, системи обчислень.

Node.js є популярним середовищем виконання, яке використовують великі та відомі компанії, такі як eBay, General Electric, GoDaddy, Microsoft, PayPal, Uber, Wikipins, Yahoo! та Yammer. Випущений за домашньою ліцензією MIT, Node.js є відкритим програмним забезпеченням, що означає, що його можна вільно використовувати, модифікувати та розповсюджувати.

Одним з інструментів, який широко використовується для розробки Node.js додатків є NPM (Node Package Manager). Це єдиний репозиторій пакетів для Node.js, який містить понад 1 мільйон пакетів для різних потреб розробників.

Крім того, Node.js підтримує велику кількість фреймворків, таких як Express.js, Hapi.js, Meteor.js, Koa.js та багато інших. Ці фреймворки допомагають розробникам створювати веб-додатки швидше та ефективніше, забезпечуючи різноманітні функції та функціональність.

У загальному, Node.js є потужним інструментом для розробки веб-додатків, який забезпечує швидкість роботи та масштабованість сервера, а також велику кількість функцій та пакетів для розробників. Він є популярним середовищем виконання, яке використовують великі та малі компанії для розробки своїх веб-

додатків та послуг.

1.5.2 Протоколи взаємодії HTTP\HTTPS

HTTP - це протокол, який дозволяє клієнтам (зазвичай, веб-браузерам) запитувати веб-сторінки з веб-серверів. Клієнти надсилають запити HTTP на веб-сервер, щоб отримати веб-сторінки, і сервери відповідають запитами HTTP. Клієнти зазвичай використовують методи GET або POST, щоб надіслати свої запити, а сервери відповідають повідомленнями про стан і надсилають запитаний ресурс. Веб-сервери за замовчуванням використовують порт TCP 80.

Під час взаємодії клієнтів і веб-серверів за допомогою протоколу HTTP використовуються методи запиту та відповіді. Клієнти, зазвичай веб-браузери, надсилають запити HTTP до веб-сервера, використовуючи методи GET або POST, наприклад GET /homepage.html. Після отримання запиту, веб-сервер надсилає відповідь HTTP, що містить повідомлення про стан, наприклад 200, якщо запит був успішним, та запитаний ресурс.

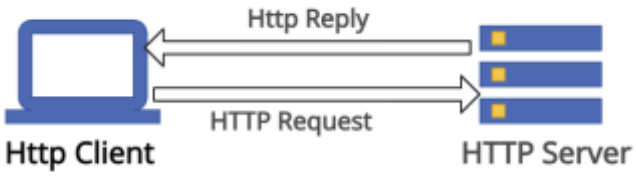


Рисунок 1.6. Схема роботи протоколу HTTP

Прикладом взаємодії між клієнтом та сервером через HTTP. Є клієнт, який хоче отримати доступ до веб-сторінки за допомогою браузера та спрямовує його на URL-адресу запиту. Веб-сервер, який розміщує цю сторінку, отримує запит та відповідає відповідним повідомленням HTTP, передаючи вміст сторінки. Зазвичай веб-сервери використовують TCP-порт 80, а якщо порт не вказаний у URL-адресі, браузер використовує цей порт за замовчуванням. Версія HTTP, яка найчастіше використовується сьогодні, є HTTP/1.1, але також доступна й підтримується більшістю браузерів новіша версія HTTP.

HTTPS (Hyper Text Transfer Protocol Secure) - це протокол, який забезпечує безпечний зв'язок між клієнтом та сервером в Інтернеті. Це досягається шляхом шифрування даних, які передаються між ними. HTTPS є захищеною версією HTTP і використовується для створення захищеного каналу в мережі Інтернет. У Інтернеті багато трафіку, який може бути доступним для хакерських атак, тому захищена передача даних є дуже важливою.

Для шифрування HTTPS використовує протокол TLS (Transport Layer Security) або попередній рівень Secure Sockets Layer (SSL). Це дозволяє шифрувати конфіденційну інформацію, таку як паролі, номери кредитних карток та інші особисті дані, що робить з'єднання безпечним. Таким чином, з'єднання з сайтами, які використовують HTTPS, стає надійним та захищеним.

URL-адреси HTTPS починаються з https замість http. Якщо ви бачите блокування праворуч від адресного рядка, це означає, що веб-сайт використовує HTTPS. Крім того, HTTPS використовує добре відомий порт TCP 443. Якщо порт не вказаний у URL-адресі, браузери автоматично використовують цей порт під час надсилання запиту HTTPS.

Усі ці заходи дозволяють користувачам Інтернету використовувати його без ризику порушення конфіденційності та захищеності своїх даних. HTTPS став стандартом безпеки в Інтернеті і відіграє важливу роль у забезпеченні безпеки на різних веб-сайтах..

1.5.3 Locators в автоматизації тестування

Locators - це інструмент, що використовується в автоматизації тестування, щоб ідентифікувати елементи на веб-сторінках. Це дозволяє тестувальникам взаємодіяти з веб-сторінками на автоматизований спосіб, що полегшує процес тестування.

Locators дозволяють знайти конкретний елемент на веб-сторінці, який потрібно перевірити або з яким потрібно взаємодіяти. Існує кілька типів локаторів, які можна використовувати в автоматизованому тестуванні, такі як:

- **ID**: Ідентифікаційний номер елемента, що знаходиться на веб-сторінці. Це

унікальний ідентифікатор для кожного елемента. Якщо на сторінці є елемент з унікальним ідентифікатором, його можна легко знайти за допомогою локатора ID.

- **Name**: Ім'я елемента, яке використовується для ідентифікації його на веб-сторінці. Локатор імені може бути використаний для знаходження елементів, що мають ім'я, яке відповідає конкретному критерію.

- **Class Name**: Назва класу, який використовується для опису елемента на веб-сторінці. Клас елемента визначає його властивості та стиль. Локатор класу може бути використаний для знаходження елементів, що мають певний клас.

- **Tag Name**: Назва тегу HTML елемента, який використовується для ідентифікації його на веб-сторінці. Наприклад, якщо на сторінці є список, тег є <ul>, тоді можна використовувати локатор тегу для знаходження списку.

- **Link Text**: Текстове посилання, яке використовується для ідентифікації посилання на веб-сторінці. Локатор текстового посилання можна використовувати для знаходження посилань на сторінці за їх текстовими назвами.

- **Partial Link Text**: Частковий текст посилання, який використовується для ідентифікації посилання на веб-сторінці. Локатор часткового тексту посилання використовується для знаходження посилань на сторінці, текст яких містить певний критерій.

- **XPath**: Універсальний шлях до елемента на веб-сторінці, який можна використовувати для ідентифікації будь-якого елемента. Якщо жоден з інших локаторів не підходить для знаходження елементу, можна використовувати локатор XPath.

Locators - це важливий інструмент для автоматизованого тестування, оскільки вони дозволяють точно ідентифікувати елементи на веб-сторінці. Правильне використання локаторів забезпечує ефективне і точне тестування веб-сторінок.

Загалом, знання і використання різних типів локаторів забезпечує ефективність і точність тестування веб-сторінок. При використанні локаторів тестувальникам потрібно враховувати унікальні властивості кожного елемента на веб-сторінці. Також, рекомендується використовувати локатори, які мають найбільш точний та швидкий пошук елементів.

1.5.4 Restful APIs

RESTful - це архітектура веб-систем, що забезпечує доступ до інформації про свої ресурси, а також можливість здійснювати дії з цими ресурсами, такі як створення нових ресурсів або зміна існуючих ресурсів. Для того, щоб API підтримували RESTful, важливо дотримуватися певних обмежень під час їх проектування. У термінах RESTful, сервер передає клієнтові подання стану запитуваного ресурсу, тобто він повертає інформацію про цей ресурс, таку як його ім'я, кількість публікацій, підписників і т.д. Наприклад, коли розробник викликає Instagram API для отримання інформації про конкретного користувача, API повертає його поточний стан з усіма деталями.

Майже всі API представляють стан у форматі JSON, хоча деякі також можуть підтримувати формати XML або HTML. Щоб взаємодіяти з API, клієнт повинен надати ідентифікатор ресурсу, який вказує на URL-адресу, а також метод HTTP, який вказує на операцію, яку потрібно виконати на цьому ресурсі. Наприклад, для отримання інформації про користувача Twitter за допомогою RESTful API, потрібно вказати URL-адресу, що ідентифікує користувача, та метод HTTP GET. Іншим прикладом є унікальний ідентифікатор користувача у Twitter, який можна отримати за допомогою її імені користувача в URL-адресі. Використання методу HTTP GET означає, що ми хочемо отримати інформацію про цього користувача.

1.6 Постановка завдань дослідження

Перед тим як розпочати проектування автоматизованої системи тестування та вибір конкретних інструментів для реалізації, було вирішено провести дослідження різних методологій та підходів до автоматизації тестування програмного забезпечення. Для ефективної розробки було сформульовано завдання з дослідження певних технологій та інструментів, які допоможуть реалізувати розроблену концепцію:

- Дослідити засоби та інструменти аналоги для автоматизації тестування.
- Дослідити інструменти та розширення для інструменту Cypress.

- Дослідити методології та патерни для Автоматизованого тестування web-сайту.
- Дослідити всі наявні можливості технології Cypress.
- Дослідити інструменти відтворення тестових сценаріїв задопомогою мови програмування JS.

2. РОЗРОБКА СТРУКТУРИ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ WEB САЙТУ

2. 1 Завдання для автоматизованого тестування web сайту

Автоматизоване тестування веб-сайтів допомагає забезпечити якість та надійність програмного забезпечення. Щоб зрозуміти, які завдання можуть бути автоматизовані в тестуванні веб-сайту, розглянемо детальніше декілька підходів та технологій, які використовуються для цього.

Один з підходів до автоматизованого тестування веб-сайтів - це використання фреймворку тестування Cypress. Цей фреймворк дозволяє розробникам тестів швидко та легко створювати тести, що відображають реальне поведінку користувачів.

Завдання, які можуть бути автоматизовані в Cypress, включають:

1. Тестування функціональності: це включає перевірку правильності відображення сторінок, роботу форм та інтерактивних елементів, валідацію даних, перевірку редіректів та інші подібні завдання.

2. Тестування продуктивності: це включає перевірку швидкості завантаження сторінок, реакції на запити користувачів, здатності сайту працювати при високому навантаженні та інші параметри продуктивності.

3. Тестування безпеки: це включає перевірку вразливостей сайту, перевірку захисту від атак, перевірку правильності роботи захисних механізмів та інші схожі завдання.

4. Тестування доступності: це включає перевірку того, як сайт працює для людей з різними видами інвалідності або тими, які використовують допоміжні технології для доступу до сайту.

5. Тестування розширення: це включає перевірку того, як сайт працює з різними браузерами, опціональними системами, екранами та іншими аспектами, що впливають на розширення функціональності сайту.

6. Тестування інтеграції: це включає перевірку того, як різні компоненти сайту працюють разом та взаємодіють між собою.

7. Тестування стійкості: це включає перевірку того, як сайт працює при несподіваних ситуаціях, таких як відключення мережі, збої сервера та інші негативні сценарії

Ці завдання можуть бути автоматизовані за допомогою Cypress, що дозволяє розробникам тестів швидко та легко створювати тести та запускати їх автоматично. Крім того, Cypress дозволяє візуалізувати результати тестів, що допомагає швидко виявляти проблеми та вносити відповідні зміни до коду.

Загалом, автоматизоване тестування веб-сайту є важливим етапом в розробці програмного забезпечення, що допомагає забезпечити його якість та надійність. Використання фреймворку Cypress дозволяє розробникам тестів швидко та ефективно створювати тести для різних завдань, що сприяє швидкому виявленню та виправленню проблем.

2.2 Моделювання об’єкта тестування

2.2.1 Діаграма прецедентів системи

Головна мета створення програмної системи полягає в тому, щоб допомогти користувачам виконувати їх повсякденні завдання. Для досягнення цієї мети, необхідно визначити вимоги до системи, але іноді користувачі можуть скласти список функцій, який не дає чіткого уявлення про те, чи зможе майбутня система задовольнити їх потреби.

Для того, щоб краще зрозуміти, як система має працювати, все частіше використовуються варіанти використання або прецеденти. Вони описують послідовність дій, які система може виконати у відповідь на зовнішні впливи користувачів або інших програмних систем. Варіанти використання дозволяють ранжувати функції системи за значимістю одержуваного результату для користувача.

Варіанти використання використовуються для визначення функціональних

вимог до системи та керують усім процесом розробки. Аналіз, проектування та тестування виконуються на основі варіантів використання. Під час аналізу та проектування вони дозволяють зрозуміти, як результати, які хоче отримати користувач, впливають на архітектуру системи та як повинні поводитися її компоненти, щоб реалізувати потрібну для користувача функціональність.

У процесі тестування варіанти використання дозволяють простіше оцінити точність реалізації вимог користувачів та дозволяють провести покрокову перевірку цих вимог.

"Стратегія використання прецедентів при визначенні вимог" означає, що крім запитання "що користувачі очікують від системи?", потрібно задавати запитання "що система повинна зробити для учасників форуму?" для пошуку функцій, які будуть корисні для багатьох користувачів, та виключення можливостей, які не допоможуть виконати повсякденні завдання користувачів.

"Діаграма варіантів використання" складається з акторів, для яких система виконує дію, та дій Use Case, які описують, що актор хоче отримати від системи. На діаграмі актори позначаються символом чоловічка, а Use Case – овалом.

Метою проекту є створення системи, яка зберігатиме дані про клієнтів та угоди з ними, а також автоматично формуватиме рахунки, які менеджер відправлятиме клієнтам. Користувачами системи будуть менеджери відділу продажів середнього підприємства. На діаграмі показані основні дії, доступні користувачеві системи керування малим і середнім бізнесом.

Рисунок 2.1 – UML Діаграма прецедентів сайту

2.3 Структура Автоматизованого тестування

2.3.1 Вибір моделі для автоматизованого тестування

Автоматизоване тестування веб-сайтів - це невід'ємна частина процесу розробки програмного забезпечення, що дозволяє перевірити функціональність, надійність та безпеку веб-додатків. Під час розробки тестів для веб-сайту важливо визначитись з патерном або шаблоном, який буде використовуватись для автоматизованого тестування.

Шаблони автоматизованого тестування допомагають зменшити кількість коду, що потрібний для написання тестів, забезпечуючи більшу ефективність та швидкість розробки. Існує багато різних патернів, які можна використовувати для автоматизованого тестування веб-сайту. Розглянемо декілька з них.

Перший патерн - Page Object. Цей патерн використовує класи для представлення веб-сторінок та їх елементів, що дозволяє легко змінювати

тестувальний код при зміні веб-сторінок. Page Object також дозволяє зберігати логіку тестування в одному місці, що зменшує кількість дублювання коду.

Другий патерн - Data Driven Testing. Цей патерн дозволяє використовувати різні набори даних для тестування одного й того ж тесту. Це забезпечує більшу покриття тестами та забезпечує більшу надійність веб-додатків.

Третій патерн - Behavior Driven Development (BDD). Цей патерн дозволяє використовувати бізнес-терміни та мову для написання тестів, що забезпечує більшу зрозумілість для бізнес-користувачів та розробників. BDD також дозволяє зосередитись на функціональності веб-додатків та створити тести, які відображають бізнес-вимоги.

Четвертий патерн - Keyword Driven Testing. Цей патерн використовує таблиці ключових слів для опису дій, які потрібно виконати для кожного тесту. Це дозволяє швидко та ефективно розробляти тестові сценарії та забезпечує легку читабельність для не-технічних користувачів.

П'ятий патерн - Hybrid Testing. Цей патерн комбінує різні підходи до автоматизованого тестування, такі як Page Object, Data Driven Testing та Keyword Driven Testing, для досягнення більшої ефективності та гнучкості.

Вибір патерну для автоматизованого тестування веб-сайту залежить від потреб розробників та специфіки проекту. Важливо вибрати той патерн, який найкраще відповідає вимогам проекту та дозволяє ефективно та швидко розробляти тести. Крім того, варто враховувати рівень складності тестових сценаріїв та наявність спеціальних фреймворків та інструментів для автоматизованого тестування, які підтримують вибраний патерн.

Узагальнюючи, вибір патерну для автоматизованого тестування веб-сайту є важливим етапом в процесі розробки тестів. Різні патерни мають свої переваги та недоліки, тому важливо вибрати той, який найкраще відповідає потребам проекту та забезпечує швидку та ефективну розробку тестів.

2.3.2 Структура патерну Page Object

Структура патерну Page Object в автоматизованому тестуванні є однією з

найважливіших концепцій в контексті тестування веб-додатків. Вона дозволяє зменшити залежність від конкретного інтерфейсу користувача, що дозволяє більш ефективно проводити тестування в різних умовах.

Основним елементом структури патерну Page Object є клас сторінки (Page Class), який відображає сторінку веб-сайту, яку ми тестуємо. У цьому класі містяться методи для доступу до елементів сторінки, наприклад, кнопок, текстових полів, чекбоксів та інших елементів інтерфейсу користувача. Також у класі можуть бути визначені тестові методи, які відповідають за перевірку різних функцій сайту.

Щоб забезпечити доступ до елементів сторінки з класу сторінки, можна використовувати локатори (Locators). Локатори є селекторами, які допомагають ідентифікувати елементи на сторінці. Локатори можуть бути виконані за допомогою різних методів, таких як XPath, CSS селекторів, ID та інших. Використання локаторів дозволяє зробити код більш структурованим та зрозумілим.

Методи (Methods) - це функції, які виконують дії з елементами сторінки. Методи можуть включати заповнення форм, натискання кнопок, встановлення чекбоксів та інші дії. Використання методів робить код більш читабельним та зменшує кількість дублювання коду.

Окрім класу сторінки, структура патерну Page Object може включати клас тестування (Test Class), який містить тестові методи, які використовують класи сторінок і методи. У тестовому класі ми можемо використовувати методи, які були визначені в класі сторінки, щоб тестувати відповідну функціональність.

Окрім основних елементів, структура патерну Page Object може включати ресурси (Resources), які є файлами, що містять різні ресурси, такі як текстові рядки, картинки, файли CSS та JavaScript, які використовуються на сторінці. Ресурси дозволяють зберігати дані в одному місці та легко їх використовувати в тестовому коді.

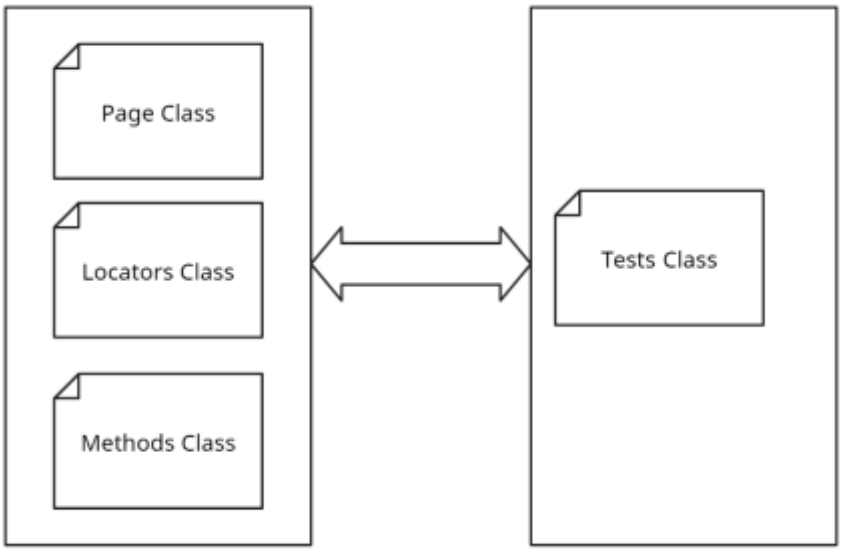


Рисунок 2.2 . – Схема патерну Page Object

Реалізація патерну Page Object дозволяє відокремити логіку тестування від логіки взаємодії з елементами сторінки. Це дозволяє зробити тестовий код більш зрозумілим та підтримуваним. Крім того, Page Object дозволяє швидко вносити зміни до тестового коду у випадку, якщо змінюється структура сторінки веб-сайту, що зменшує час на розробку та підтримку тестів.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ДОДАТКУ

3.1 Набір інструментів використаних для розробки

Мова JavaScript була обрана для автоматизації тестування сайту з кількох причин. По-перше, JavaScript є однією з найбільш популярних мов програмування, що використовуються веб-розробниками, тому є багато ресурсів та документації для вивчення мови та розв'язання проблем. Крім того, JavaScript є мовою скриптів, що дозволяє тестувальникам легко вбудовувати тестові скрипти без необхідності встановлення додаткового програмного забезпечення.

Багато інструментів автоматизації тестування, таких як Selenium та Cypress, використовують JavaScript як основну мову програмування. Використання однієї мови програмування для автоматизації тестування та розробки допомагає зменшити пороговий бар'єр для вхідного рівня тестувальників, оскільки вони вже знайомі з мовою та можуть використовувати свої знання для автоматизації тестування.

Нарешті, JavaScript є мовою програмування на стороні клієнта, що дозволяє тестувати функціональність веб-додатків безпосередньо в браузері. Це дуже зручно, оскільки веб-додатки можуть бути дуже складними та містити багато функцій, які можуть бути важко протестувати безпосередньо з коду.

Таким чином, використання JavaScript для автоматизації тестування є розумним вибором, оскільки він дозволяє розробникам легко вбудовувати тестові скрипти, може бути використаний для розробки та автоматизації тестування, та дозволяє тестувати функціональність веб-додатків безпосередньо в браузері.

В якості інтегрованого середовища розробника було вибрано Visual Studio Code, оскільки вона першочергово була розроблена для написання коду програмз використанням веб технологій та мов JS, HTML, CSS та має ряд вагомих переваг над іншими IDE, а саме (Рис 3.1.):

- Зручний користувацький інтерфейс;
- Підтримка багатьох мов програмування з коробки – JS, Python, TypeScript, C#;

- Великий набір розширень та плагінів;
- Інтеграція з системою контроля версій GIT;

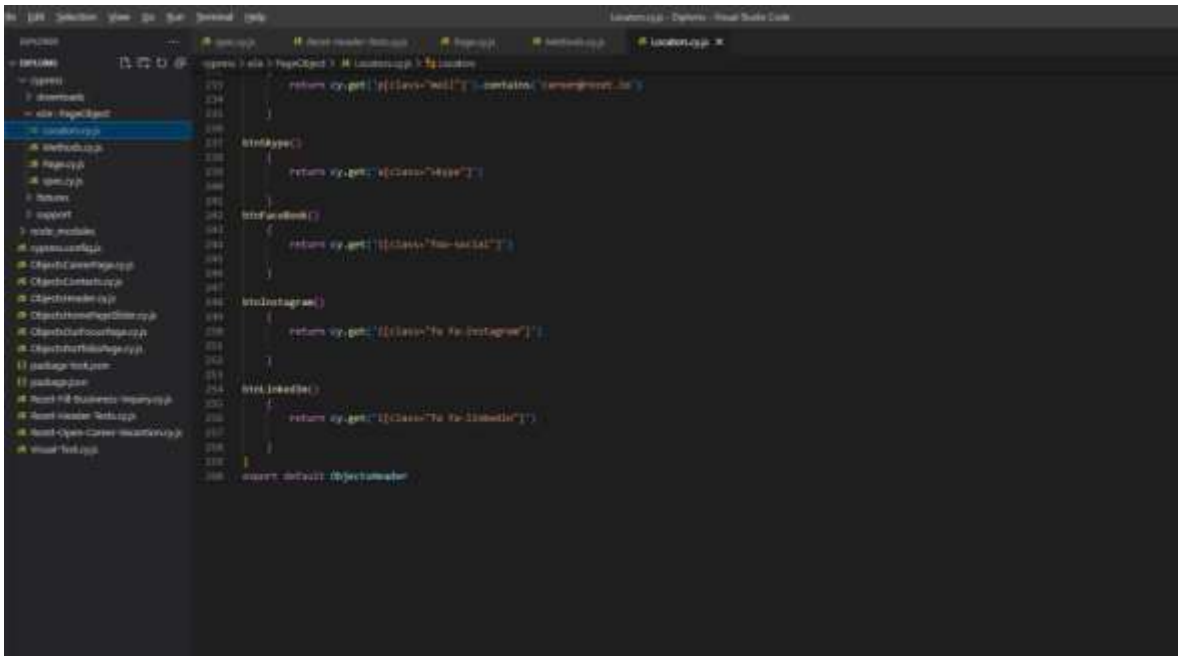


Рисунок 3.1. – Visual Studio Code

Для написання тестової документації було використано TestRail TestRail - це веб-додаток для управління тестами і тест-кейсами, який дозволяє тестувальникам та іншим учасникам проекту легко створювати, організовувати та виконувати тести. TestRail дозволяє створювати тестові плани, які включають набір тест-кейсів, що дозволяє учасникам проекту керувати процесом тестування і визначати прогрес тестування.

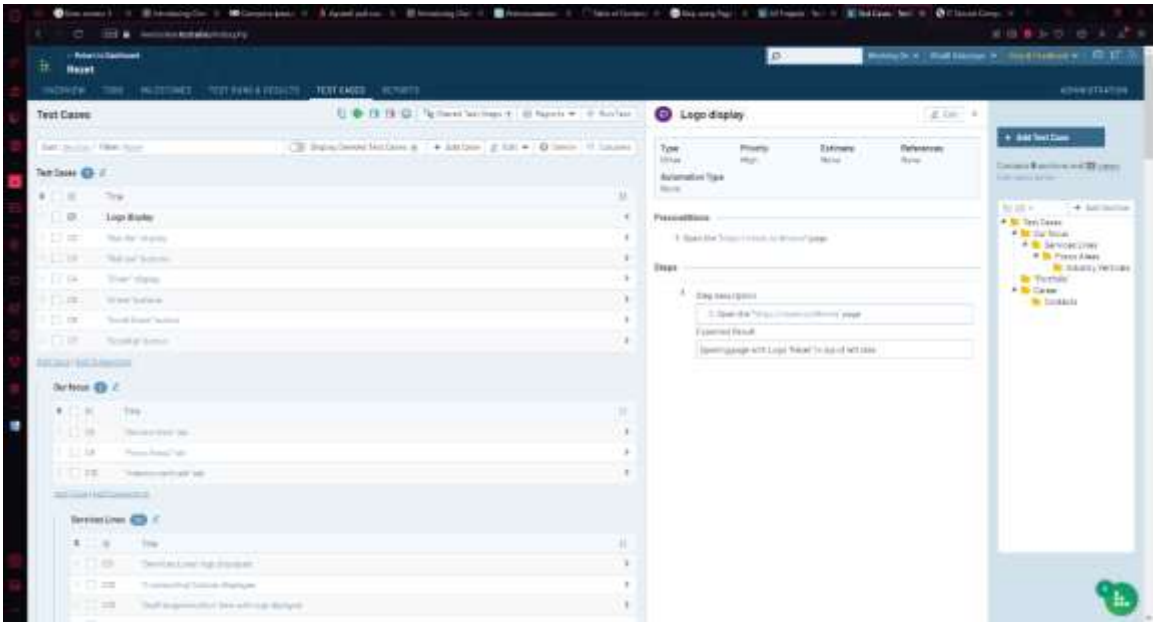


Рисунок 3.2. – TestRail

Додаток TestRail також надає зручний інтерфейс для виконання тестів, стеження за їх станом та відстеження результатів. Він підтримує інтеграцію з іншими інструментами для тестування, такими як Selenium, JIRA, Jenkins та інші.

TestRail дозволяє створювати звіти про результати тестування, що допомагає учасникам проекту аналізувати та розуміти результати тестування та приймати рішення щодо подальшої роботи над проектом.

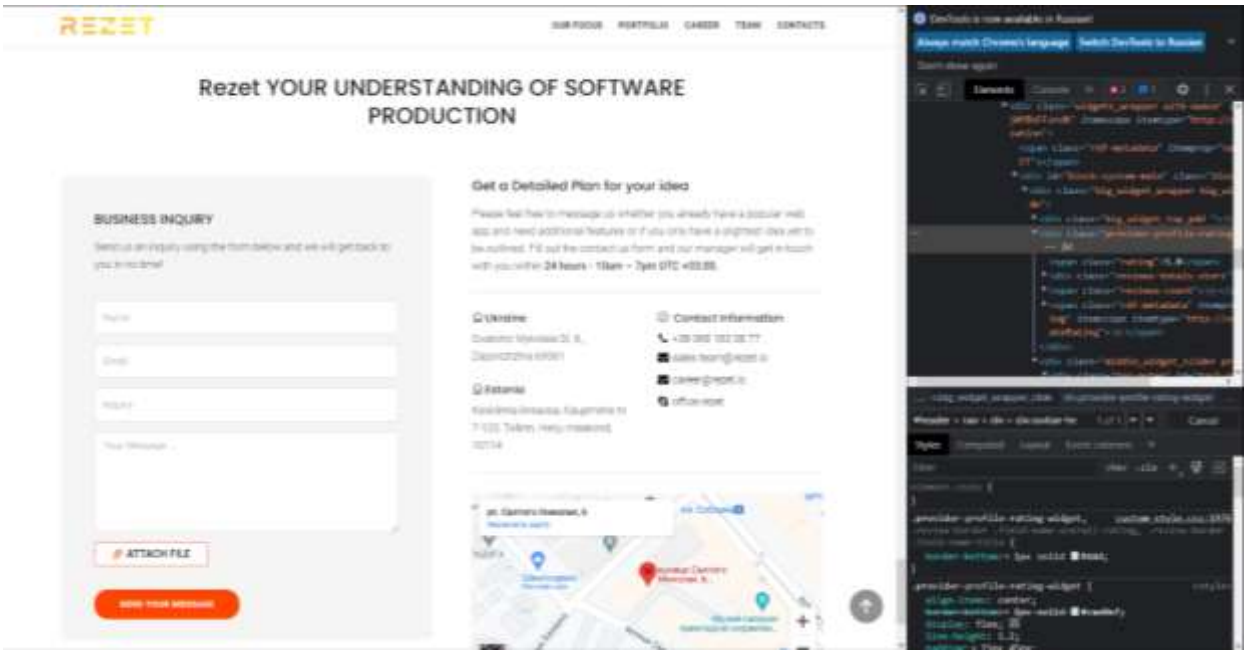


Рисунок 3.3. – Chrome dev tools .

Chrome DevTools - це інструмент, що надається браузером Google Chrome для розробників, який дозволяє аналізувати та досліджувати веб-сайти. Цей інструмент може бути корисним для автоматизації тестування в Cypress, оскільки він надає можливість отримати доступ до різних елементів сторінки, а також додаткової інформації, що допоможе в процесі написання тестів. (Рис. 3.3.)

Для розробки десктоп додатку було використано Electron Js. Electron Js - це фреймворк для створення десктоп додатків, який базується на веб-технологіях. Він дозволяє розробникам використовувати HTML, CSS та JavaScript для створення десктоп додатків на різних операційних системах, включаючи Windows, macOS та Linux.

Один з головних переваг використання Electron Js полягає в тому, що він дозволяє розробникам використовувати один і той же код для створення додатків на різних платформах. Це зменшує витрати на розробку та тестування додатків, а також дозволяє швидко розгортати додатки на різних платформах.

Ще однією важливою перевагою Electron Js є те, що він дозволяє розробникам створювати додатки з високою продуктивністю та швидкодією. Тому він часто використовується для розробки додатків, які вимагають високої

швидкодії та стабільності, таких як редактори коду, музичні плеєри, графічні редактори та інші.

Ще однією важливою особливістю Electron Js є те, що він має велику спільноту розробників та багато готових модулів, які можна використовувати для розробки додатків. Це дозволяє розробникам швидко створювати додатки з різноманітними функціями та можливостями. Наприклад, для створення крос-платформового редактора коду можна використовувати готові модулі для роботи з текстом та редагування коду, а для створення музичного плеєра - модулі для відтворення аудіо-файлів та роботи зі звуком.

Крім того, Electron Js дозволяє розробникам легко налаштовувати та розгортати додатки для різних платформ. Наприклад, для розгортання додатку на Windows можна використовувати готові інсталяційні файли, а для macOS - пакети для розгортання та встановлення додатку.

Однією з головних переваг використання Electron Js є те, що він дозволяє розробникам створювати додатки з використанням сучасних веб-технологій, таких як React, Angular та Vue.js. Це дозволяє розроблювати додатки з використанням знайомих технологій та зменшує час на навчання нових технологій.

Також варто відзначити, що Electron Js підтримує розширення та плагіни, які дозволяють розширювати можливості додатків. Наприклад, для розширення можливостей редактора коду можна використовувати різноманітні плагіни для підсвічування синтаксису, автодоповнення коду та інші.

Загалом, Electron Js є потужним фреймворком для розробки десктоп додатків, який дозволяє розробникам використовувати веб-технології для створення додатків на різних платформах. Його висока продуктивність та швидкодія, а також велика спільнота розробників та багато готових модулів роблять його ідеальним вибором для розробки десктоп додатків.

3.3 Написання тестової документації

Тестова документація є важливою частиною будь-якого проекту, включаючи дипломну роботу на тему "Автоматизація тестування сайту за допомогою Cypress".

Вона допомагає зберегти інформацію про тестовий план, тестові кейси та їх результати.

У цьому розділі дипломної роботи наведені основні вимоги до тестової документації та рекомендації щодо її написання. Керуючись цими рекомендаціями, ви зможете забезпечити якість вашого програмного продукту, підвищити його надійність та ефективність.

Тестова документація повинна включати наступні елементи:

- Опис функціональності сайту, який тестується;
- Список тестових кейсів з описом кожного кейсу;
- Інформацію про середовище, на якому виконувалися тести;
- Результати тестування, включаючи опис помилок, які були виявлені під час тестування;
- Висновки та рекомендації щодо подальшого тестування та вдосконалення сайту.

Опис функціональності сайту має бути написаний детально та зрозуміло для кожного члена команди. Список тестових кейсів повинен бути повний та змістовний. Для кращої структуризації тестових кейсів, їх можна розділити на групи за певними критеріями, наприклад, за функціональністю або за пріоритетом.

Інформація про середовище, на якому виконувалися тести, повинна бути детально описана, включаючи версії програмного забезпечення та операційної системи. Результати тестування мають бути оформлені в зручному для аналізу форматі, такому як таблиці або графіки.

При описі помилок, необхідно зазначати не тільки їх опис, а й причину виникнення та можливі шляхи виправлення. Висновки та рекомендації щодо подальшого тестування та вдосконалення сайту мають бути зроблені детальними та конкретними.

Нижче наведені деякі рекомендації щодо написання тестової документації:

- Документація повинна бути написана на зрозумілій для всіх мові;
- Кожен тестовий кейс повинен бути описаний докладно і точно, вказуючи очікувані результати;

- Для зручності, тестові кейси можуть бути розділені на окремі групи;
- Результати тестування повинні бути представлені в зручному для аналізу форматі, наприклад, в таблицях або графіках;
- При описі помилок необхідно зазначати їх причини та можливі способи виправлення;
- Висновки та рекомендації повинні бути зроблені детальними та конкретними.

Пам'ятайте, що правильно написана тестова документація є важливою складовою успішного проекту. Вона допомагає виявити проблеми та помилки ще на ранніх етапах розробки та забезпечує якість продукту на високому рівні.

Для того, щоб тестова документація була якісною та дієвою, важливо пам'ятати про кілька ключових моментів. Перш за все, слід планувати тестування відразу після розробки функціоналу, що дозволить більш детально і точно відобразити всі нюанси та виявити помилки. Другим важливим аспектом є правильне описання тестових кейсів. Для цього потрібно бути максимально детальним та точним у визначенні критеріїв успішності ключових функціональних блоків.

Також необхідно бути уважним при написанні висновків та рекомендацій. Вони повинні бути чіткими та конкретними, а не загальними та неопределеними. Крім того, важливо використовувати різні методики тестування для отримання максимально точних результатів.

Загалом, написання тестової документації є важливим етапом в розробці будь-якого програмного продукту. Це дозволяє забезпечити високу якість продукту та виявити помилки на ранніх етапах розробки, що зменшує витрати часу та коштів на подальшу розробку та тестування.

Smoke tests - це перші тести, які проводяться для перевірки основної функціональності програмного продукту після внесення змін в код. Вони допомагають швидко виявити критичні проблеми в системі, такі як помилки під час завантаження, невідповідність веб-сторінок до дизайну, несправні функціональні можливості тощо.

Smoke тести повинні бути написані для перевірки тих функцій системи, які є найбільш критичними та використовуються найчастіше. Це можуть бути, наприклад, основні можливості входу в систему, навігація по веб-сторінках, пошук, додавання і редагування даних, виведення результатів тощо.

При написанні тестів слід пам'ятати, що вони повинні бути максимально простими та швидкими, щоб вони могли бути виконані якнайшвидше і допомогти виявити проблеми раніше.

Автоматизоване тестування є необхідною складовою процесу розробки програмного забезпечення, яке дозволяє забезпечити ефективність та якість продукту. Одним з найважливіших етапів в цьому процесі є написання тест-кейсів, які дозволяють протестувати різноманітні функції продукту та виявляти помилки на ранніх етапах розробки.

Виходячи з вище описаних правил і рекомендацій по створенню тестової документації, мною було створено 122 smoke теста для сайту компанії Rezet. Це було зроблено з метою забезпечення повноти тестування та виявлення можливих помилок. Крім того, застосування автоматизованих тестів дозволяє збільшити швидкість тестування та зменшити час витрачений на ручне тестування.

В результаті, створення тест-кейсів є важливим етапом в автоматизованому тестуванні та дозволяє забезпечити високу якість продукту та задовольнення потреб користувачів.

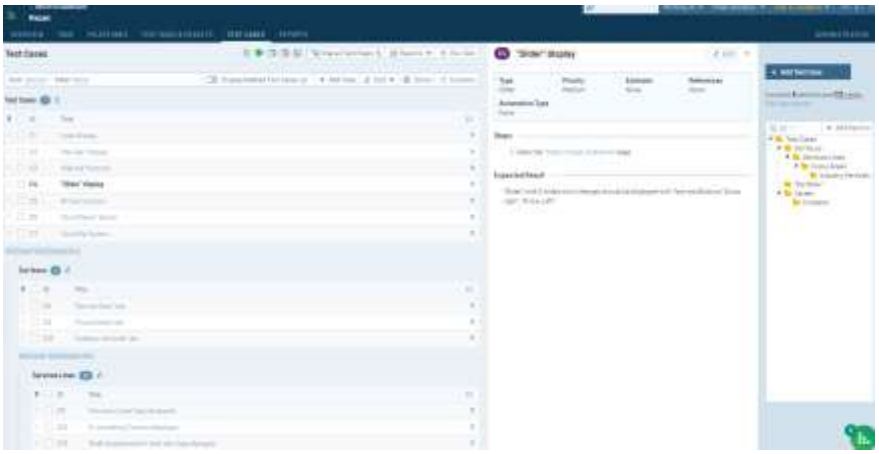


Рисунок. 3.4. – Приклад тест кейса

3.3 Створення архітектури за допомогою патерну Page Object

В автоматизованому тестуванні патерни допомагають забезпечити легкість та ефективність тестування. Один з найбільш популярних патернів для автоматизації тестування - це патерн Page Object, який використовується для кращого управління тестовими скриптами. В даному випадку, я вибрав патерн Page Object який, на мою думку, забезпечить лаконічність та легкість у використанні в проекті у майбутньому для написаних авто тестів.

Щоб реалізувати патерн Page Object, я створив кілька файлів з класами в середині. Перший файл - Locators.cy.js - містить всі наявні елементи на сайті з вже написаними локаторами для них. Другий файл - Methods.cy.js - містить часті у використанні методи, щоб уникнути зайвого копіювання коду. Також будуть файли з автотестами, які використовують класи з попередніх файлів.

Щоб автотести могли бачити інформацію з інших класів у Cypress, потрібно виконати наступні кроки:

- Створити клас, який буде відповідати сторінці чи компоненту сайту. Наприклад, файл з назвою LoginPage.js з класом LoginPage.
- Імпортувати клас у тестовий файл, де він буде використовуватися. Наприклад, у файлі login.spec.js.
- У цьому прикладі клас LoginPage імпортується з файлу ../support/pages/LoginPage, а його екземпляр loginPage створюється у функції beforeEach() перед кожним тестом.

За допомогою Page Object підходу, ви можете зменшити кількість дублювання коду, зберегти час на написання автоматизованих тестів та отримати більш розумний та легкий управління тестовими скриптами. Використання класів з інших файлів у Cypress дозволяє створювати більш організовані та ефективні автоматизовані тести з використанням Page Object підходу. Також, якщо ви хочете додати до свого проекту Page Object підхід, вам знадобиться додатковий час на створення та оновлення класів, але це забезпечить більш просту та зручну

структуру коду.

3.4 Розробка автоматизованого тестування для сайту

При розробці автоматизованого тестування сайту за допомогою Cypress, можна використовувати більше методів, які дозволяють забезпечити повну автоматизацію тестування. Нижче перераховані додаткові методи, які можуть бути використані:

- cy.clearCookies() - цей метод використовується для видалення всіх cookies з домену. Використання цього методу допомагає забезпечити більш точне тестування, оскільки дозволяє очистити всі дані, які можуть впливати на результати тестування.
- cy.clearLocalStorage() - цей метод використовується для видалення всіх даних з localStorage. Використання цього методу допомагає забезпечити більш точне тестування, оскільки дозволяє очистити всі дані, які можуть впливати на результати тестування.
- cy.getCookie() - цей метод дозволяє отримати cookie по імені. Використання цього методу допомагає забезпечити точне тестування, оскільки дозволяє перевірити, що cookie було встановлено правильно.
- cy.setCookie() - цей метод дозволяє встановити cookie. Використання цього методу допомагає забезпечити точне тестування, оскільки дозволяє перевірити, що cookie було встановлено правильно.
- cy.scrollTo() - цей метод використовується для прокрутки до потрібного елемента на сторінці. Використання цього методу допомагає забезпечити точне тестування, оскільки дозволяє перевірити, що елемент, до якого потрібно прокрутити сторінку, існує на сторінці.
- cy.viewport() - цей метод дозволяє змінити розмір вікна браузера. Використання цього методу допомагає забезпечити точне тестування, оскільки дозволяє перевірити, що сайт працює коректно на різних розмірах екрану.
- cy.intercept() - цей метод дозволяє перехоплювати запити до сервера та

змінювати їх. Використання цього методу допомагає забезпечити точне тестування, оскільки дозволяє змінити поведінку сервера та перевірити, як сайт реагує на різні сценарії.

- cy.route() - цей метод використовується для зміни поведінки запитів на сторінці. Використання цього методу допомагає забезпечити точне тестування, оскільки дозволяє змінити поведінку сайту та перевірити, як він реагує на різні сценарії.

Ці методи дозволяють створювати більш складні тести для сайту та забезпечувати повну автоматизацію процесу тестування за допомогою Cypress. Використання цих методів дозволяє не тільки зменшити витрати на ручне тестування, покращити якість тестування та забезпечити більш швидку роботу, але і дозволяє забезпечити більш глибоку та точну перевірку функціональності сайту. Наприклад, використання методу cy.intercept() дозволяє змінювати поведінку сервера та перевіряти, як сайт реагує на різні сценарії, що допомагає забезпечити більш точне тестування та виявити помилки, які можуть бути пропущені за звичайних умов.

```
import Locators from './Locators.cy.js'

describe('Contacts fill Bussines inqury',  () => {

const locators =new Locators()
const Name = 'Vitalii'
const Email = 'EmailRezet@rezet.io'
const Inquiry= 'Test'
const yourMessage = 'TEST1234'

 it('Positive test', () => {

   cy.visit('https://rezet.io/')

   locators.inputFieldName().type(Name)
   locators.inputFieldName().should('have.value',Name)

   locators.inputFieldEmail().type(Email)
   locators.inputFieldEmail().should('have.value',Email)

   locators.inputFieldInquiry().type(Inquiry)
   locators.inputFieldInquiry().should('have.value',Inquiry)

   locators.inputFieldYourMessage().type(yourMessage)
   locators.inputFieldYourMessage().should('have.value','TEST1234')

   cy.reload()

  // locators.btnSendMessage().click()

 })

 it('Negative Test', () => {
   cy.visit('https://rezet.io/')
locators.btnSendMessage().click()

cy.get('div[class="error"]').contains('Please enter your name').should('be.visible')

 })

 it('Test Email', () => {
   cy.visit('https://rezet.io/')

   locators.inputFieldEmail().type(Email)
   locators.inputFieldEmail().should('have.value',Email)
 })
```

Рисунок 3.5. – Приклад одного з автотестів

Зображено, що з початку імпортуються дані з файлу з локаторами, який має назву “Locators” для цього була використанна функція “import Locators from ‘./Locators.cy.js”

Наступним кроком було використана функція describe()

describe() - це функція в тестовому фреймворку Cypress, яка дає змогу групувати тести в логічні блоки. Далі створюємо змінні які ми будемо надалі використовувати у тестах для цього використовуєммо функціх Const так як надаллі данні в змінніі змінюватись не будуть. Щоб створити перший авто тест було використано функцію It() вона допоже розділити тести між собою та спростити їх аналіз в подальшому.

І так ми зрозумили як оформити свій файл автоматизованого тестування і далі можему приступити для написання самих автотестів. Щоб вікрити сайт потрібно використати фунцію cy.visit(‘https://rezet.io’)

Далі нам потрібно перейти на елемент котрий нам потрібно для цього визиваємо його через фунцію locators.inputFieldName() де Locators це клас в якому знаходиться елемент а inputFieldName() назва функції яка розташована в іншому файлі з локаторами і займається пошуком елементу через функцію cy.get('input[class="form-control name"]') Далі ми вказуємо що потрібно записати данні з змінної Name у елемент котрий ми знайшли. Для цього використаємо функцію .type() яка і буде цім займатись. Щоб впевнитись що данні були внесені використаємо таку функцію : locators.inputFieldName().should('have.value',Name)

Де .should(‘have.value’) вказує що вибраний елемент повинен мати данні с змінної Name.

Наприкінці кінці потрібно позакривати всі дужки і цей тест кейс уже готовий до використання.

4. ПРИКЛАДИ ВИКОРИСТАННЯ ТА ТЕСТУВАННЯ СИСТЕМИ

4.1 Приклад запуску одного з набіру тест кейсів

Щоб ініціалізувати автоматизоване тестування потрібно спочтаку запустити додаток і в інтерфейсі додадку (Рис. 4.1.). В інтерфейсі продемонстровано всі наявні тести які можна запустити як тест. Щоб їх запустити потрібно вибрати потрібний файл і запустит його клікнувши по кнопці “Run test” для потрібного тесту . Після цього запуститься скрипт який і буде перевіряти сайт

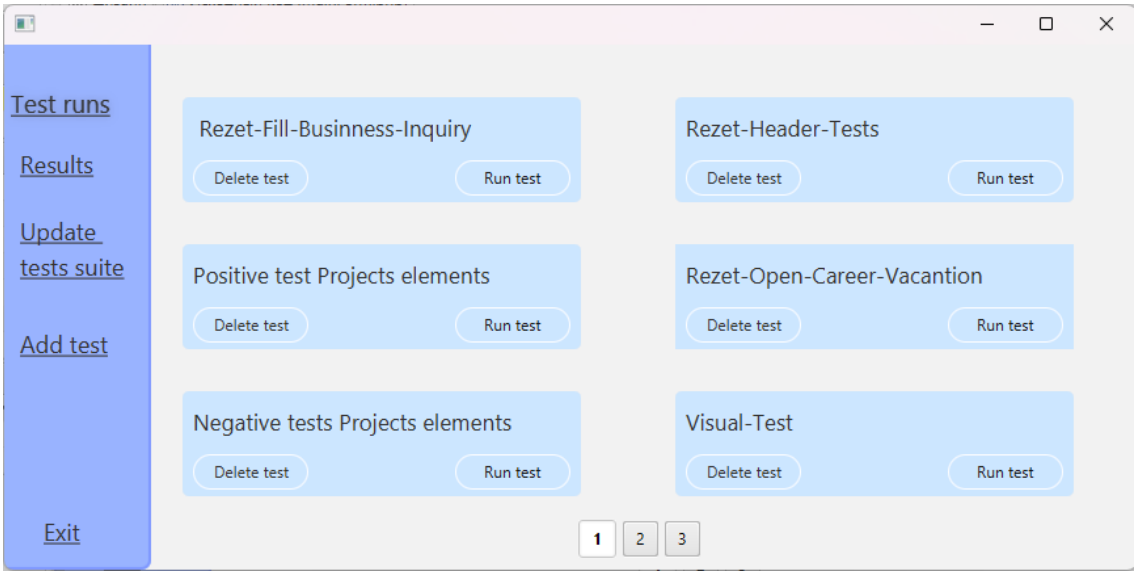


Рисунок. 4.1. – Головний інтерфейс додатку

Після того як користувач вибрав потрібний йому тест. Відкривається термінал з відображенням процесу проходження авто тесту.

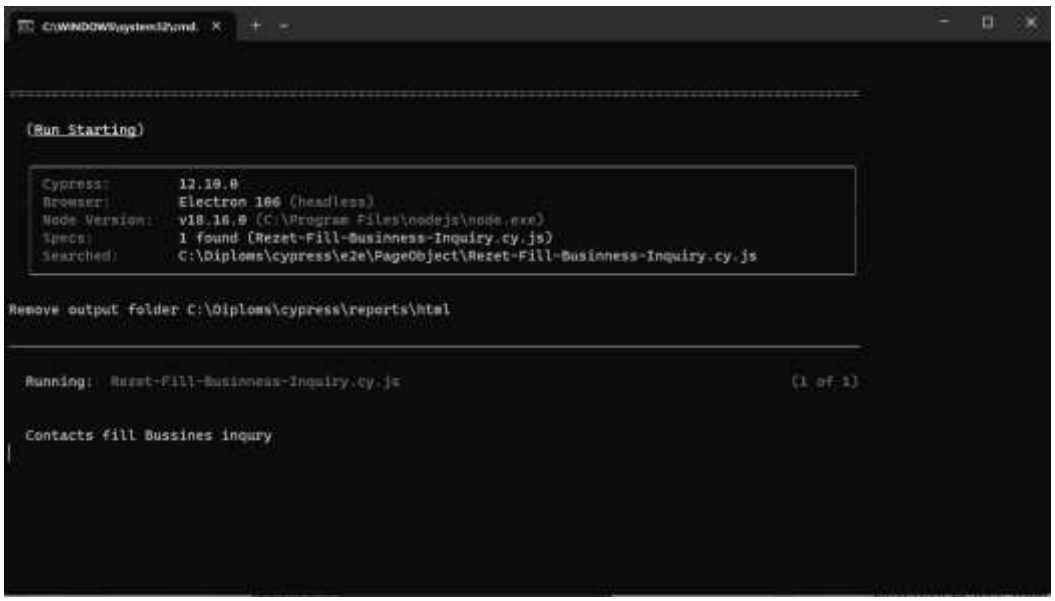


Рисунок 4.2. – Процес проходження авто тестів

Після того як Cypress успішно запустився та завершив потрібний авто тест, користувач може відкрити результат тесту натиснувши на кнопку “Results”

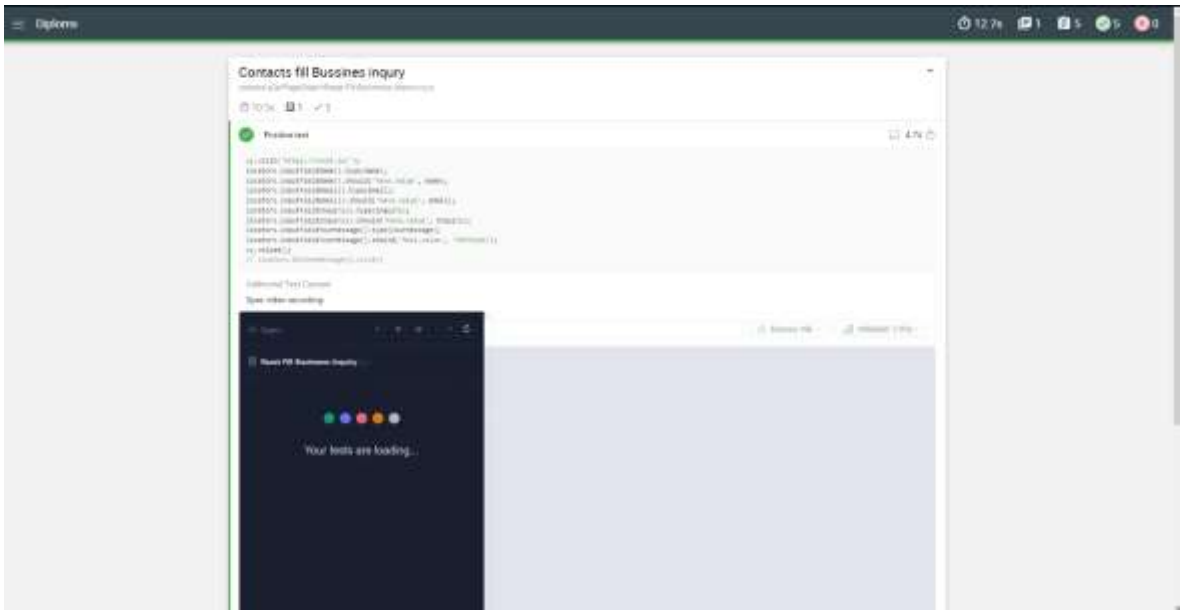


Рисунок 4.3. – Результат пройденого тесту

На рисунку 4.3 зображено список відтворених тестів. Для зручності користувача, успішно пройдені тести позначені зеленою галочкою, а невдалі - червоним хрестиком. Це дозволяє швидко зорієнтуватись у результаті

тестування і побачити, які тести потребують уваги. Якщо тест завершився невдало, користувач може отримати більш детальну інформацію про помилку, натиснувши на червоний хрестик.

Додатково до цього, в інтерфейсі додано запис проходження тестів щоб користувач мав змогу побачити на якому менті сталась невдача а бо перевірити коректність проходження тесту.

4.2 Набір тестових сценаріїв для забезпечення якості продукту

Перевірка автоматизованих тестів за допомогою ручного тестування - це важливий етап у процесі тестування, який дозволяє забезпечити високу якість програмного забезпечення. Оскільки автоматизовані тести можуть мати свої обмеження та не здатні виявити всі можливі проблеми, перевірка за допомогою ручного тестування допомагає виявляти проблеми, які можуть бути пропущені в процесі автоматизованого тестування.

Під час перевірки автоматизованих тестів за допомогою ручного тестування, тестер виконує автоматизовані тести та перевіряє результати вручну. Це може включати перевірку логів, виведення повідомлень про помилки та інші засоби, що допомагають ідентифікувати можливі проблеми. Таким чином, тестер може більш детально ознайомитись з функціональністю тестування та зрозуміти, як вона працює в реальному середовищі.

Однак, перевірка автоматизованих тестів за допомогою ручного тестування може зайняти додатковий час та зусилля. Тому важливо правильно розподілити час між автоматизованим тестуванням та ручним тестуванням. Крім того, важливо правильно вибрати тести, які будуть перевірятись за допомогою ручного тестування. Також, важливо забезпечити, щоб тестери мали достатньо ресурсів та інструментів для виконання своєї роботи.

Перевірка автоматизованих тестів за допомогою ручного тестування дозволяє покращити якість автоматизованих тестів та зменшити кількість пропущених проблем. Крім того, це дозволяє тестувальникам більш детально

ознайомитись з функціональністю тестування та зрозуміти, як вона працює в реальному середовищі. Також, це допомагає збільшити кількість тестів, які можуть бути автоматизовані в майбутньому, та зменшити кількість помилок, які можуть з'явитися під час автоматизованого тестування.

У підсумку, перевірка автоматизованих тестів за допомогою ручного тестування є важливим етапом у процесі тестування програмного забезпечення. Цей процес допомагає забезпечити високу якість програмного забезпечення та зменшити кількість помилок. Важливо правильно вибрати тести, які будуть перевірятись, та розподілити час між автоматизованим тестуванням та ручним тестуванням. Це дозволить забезпечити ефективну роботу тестерів та підвищити якість програмного забезпечення.

Так як ми уже написали тестові сценарії в попередніх пунктах ми можемо використовувати їх так як ці тести можуть використовуватись як у автоматизованому тестування так і у мануальному без ніяких проблем.

Тому використовуючи ці тестові сценарії я провів тестування автотестів і співставив їх результати. Вони повністю пройшли перевірку що може означати про успішність і якість розробленого автоматизованого тестування.

ВИСНОВКИ

У результаті виконання даної дипломної роботи досліджено та розроблено комплексну систему автоматизованого тестування веб-сайту компанії Rezet.

1. Проведено аналіз архітектури існуючих інструментів автоматизації тестування. Було проведено дослідження різних інструментів тестування, враховуючи можливості кожного з них, вимоги проекту. Вибрано такі інструменти, як Selenium, TestNG, Mocha та інші. Та встановлено, що Cypress є найбільш підходящим для даного проекту інструментом.

2. Було порівняно інструменти автоматизованого тестування та виявлено переваги та недоліки кожного з них. Наприклад, Selenium є дуже популярним та має велику спільноту користувачів, але має деякі недоліки, такі як складність налаштування, довгий час виконання тестів та складність в роботі з деякими типами веб-елементів. TestNG є простим та зручним інструментом, який дозволяє виконувати тести на основі анотацій, але не має вбудованої підтримки для запуску тестів у браузері. Mocha є одним з найпопулярніших інструментів для тестування JavaScript, але не має вбудованої підтримки для автоматизованого тестування веб-сайтів. Під час аналізу було визначено, що Cypress є найбільш підходящим для даного проекту інструментом, оскільки він дозволяє швидко та зручно писати тести, має простий та зрозумілий інтерфейс та забезпечує високу швидкість виконання тестів.

3. Розроблено програмний продукт, що забезпечує використання автоматизоване тестування сайту на базі Cypress. Створено різноманітні тестові скрипти, які перевіряли функціонал сайту з різних сторінок та різними способами. Для забезпечення максимальної ефективності тестування, проведено ряд налаштувань середовища, де проводилось тестування. Результати тестування були збережені та оброблені, щоб знайти та виправити всі помилки та проблеми з роботою сайту.

4. Під час тестування програмного забезпечення перевірено роботу всіх функцій сайту та виявлено деякі помилки, які були виправлені. Крім того,

проведено тестування на різних конфігураціях та середовищах, щоб забезпечити максимальну надійність та стабільність роботи програмного продукту. Результати тестування були детально проаналізовані та документовані, щоб забезпечити максимальну якість та надійність роботи сайту компанії Rezet.

Під час розробки було використано новітні технології та методи, що дозволило створити зручний та інтуїтивно зрозумілий інтерфейс користувача та забезпечити високу швидкість та точність тестування.

ПЕРЕЛІК ПОСИЛАНЬ

1. Why Cypress?[Електронний ресурс] // Cypress Documentation. – 2023. – Режим доступу до ресурсу: https://docs.cypress.io/guides/overview/why-cypress.

2. Selenium [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://www.selenium.dev/documentation/.

3. TestNG [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://testng.org/doc/.

4. Mocha [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://mochajs.org.

5. Сазерленд С. "Test Driven Development for Embedded C" ("Розробка програм на C з тестуванням на перший зіткнення") / С. Сазерленд, Д. Шора., 2011. – (Addison-Wesley Professional).

6. Elfriede Dustin, Jeff Rashka, John Paul. Automated Software Testing: Introduction, Management, and Performance: Introduction, Management, and Performance./- 2007 – с. 277–295.

7. Roy Sutton. Enyo: Up and Running: Build Native-Quality Cross-Platform JavaScript Apps. /- “O'Reilly Media”, 2015 – с. 87.

8. Mike Richardson, Ruby Leonard, Sam Amundsen. RESTful Web APIs./- “O'Reilly Media”, 2013 – с. 357 – 363.

9. Eric Elliott. Programming JavaScript Applications: Robust Web Architecture with Node, HTML5, and Moderns JS Libraries. -/ “O’Reilly Media”, 2013 – с. 134– 177.

10. Laiza Krispin, Janette Gregory. Agile Testing: A Practical Guide for Testers andAgile Teams. /- 2010 – с. 215–247.

11. Канер Кем, Фолк Джек, Нгуен Енг Кек. Тестування програмного забезпечення. Фундаментальні концепції менеджменту бізнес-додатків./-2001 – с. 344–375.

12. W Hetzel. Complete Guide to Software Testing (2e)./- “QED

Information Sciences: Wellesley MA", 1993 – с. 115–121.

13. A. Spillner, T. Linz, H. Schaefer. Software Testing Foundations (4e)./- "Rocky Nook: San Rafael CA", 2014 – с. 223–229.

14. ISTQB [Електронний ресурс] - Режим доступу до ресурсу: https://www.istqb.org/.

15. Software Testing Help [Електронний ресурс] – Режим доступу до ресурсу: https://www.softwaretestinghelp.com/.

16. Rex Black. Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing (3e)./- 2012 – с. 101–121.

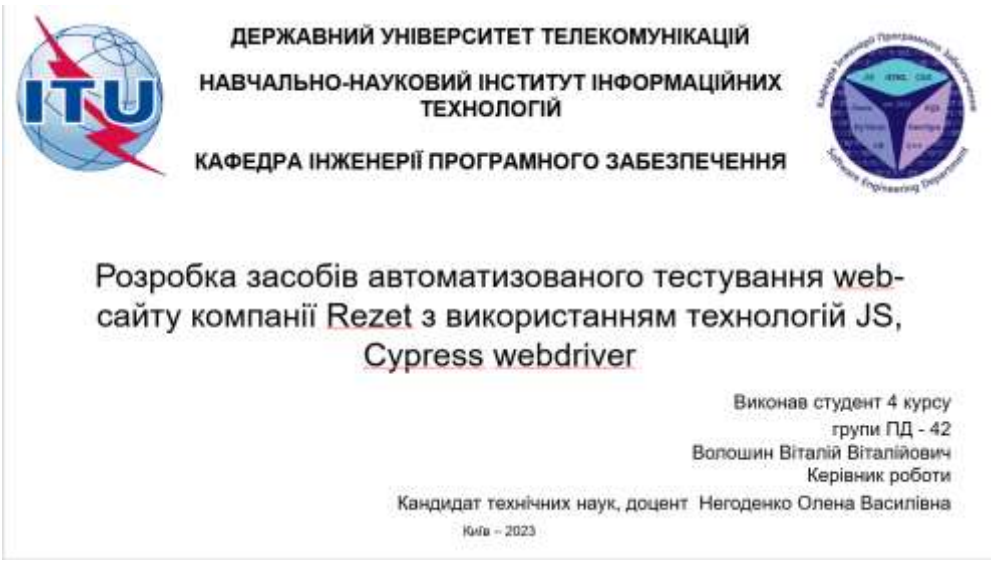
17. Lisa Crispin, Janet Gregory. More Agile Testing: Learning Journeys for the Whole Team. – с. 152–182.

18. Alan Richardson. Selenium WebDriver with Java: Basics to Advanced, 2nd Edition. – с. 352–382.

19. A. Spillner, T. Linz, H. Schaefer. Software Testing Foundations (4e)./- "Rocky Nook: San Rafael CA", 2014 – с. 223–229.

20. Laiza Krispin, Janette Gregory. Agile Testing: A Practical Guide for Testers and Agile Teams. – с. 210–250.

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ



ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка засобів автоматизованого тестування web-сайту компанії Rezet з використанням технологій JS, Cypress webdriver

Виконав студент 4 курсу
групи ПД - 42
Волошин Віталій Віталійович
Керівник роботи
Кандидат технічних наук, доцент Негоденко Олена Василівна

Київ – 2023

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

1. **Мета роботи** - спрощення процесу автоматизації тестування web-сайту компанії Rezet за рахунок використанням технологій Cypress та JavaScript.
2. **Об'єкт дослідження** - процес автоматизації тестування web-сайту.
3. **Предмет дослідження** - засоби та технології для автоматизованого тестування web-сайту.

2

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Проаналізувати архітектуру існуючих інструментів автоматизації тестування.
2. Порівняти існуючі інструменти автоматизованого тестування для виявлення їх переваг та недоліків.
3. Розробити програмне забезпечення для автоматизованого тестування сайту компанії Rezet за допомогою Cypress.
4. Провести тестування програмного забезпечення, щоб переконатися в його працездатності та відповідності вимогам.

3

АНАЛІЗ АНАЛОГІВ

Засіб	Користувацький інтерфейс	Генерація результатів	Відеозапис проходження автотестів	Інтеграція з сторонніми додатками
Cypress	-	-	+	+
Selenium	-	+	-	+
Codefuse	+	-	-	-
Mocha	-	+	-	+
Розроблений застосунок	+	+	+	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

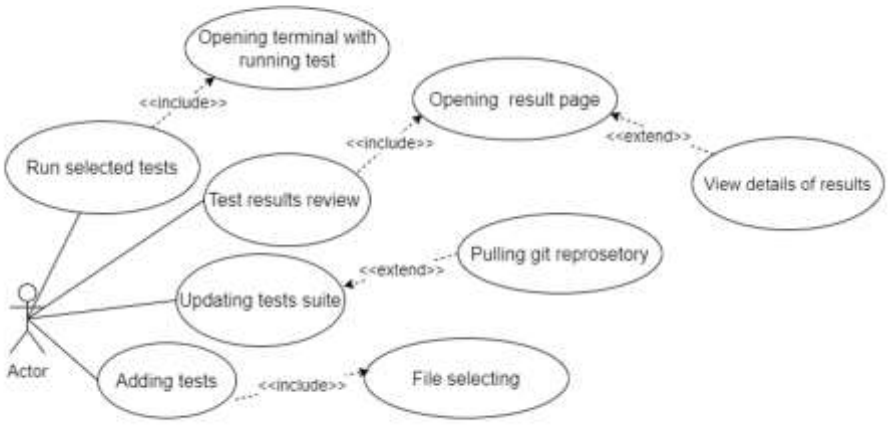
1. Виконання авто-тестів: програмне забезпечення повинно мати можливість автоматично виконувати тести, що були створені.
2. Можливість легко розширювати та підтримувати авто тести.
3. Програмне забезпечення повинно мати можливість перевіряти результати тестування та генерувати звіти про успішність/невдачі тестів.
4. Програмне забезпечення повинно підтримувати різні типи тестів, такі як функціональні, інтеграційні, модульні та інші.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

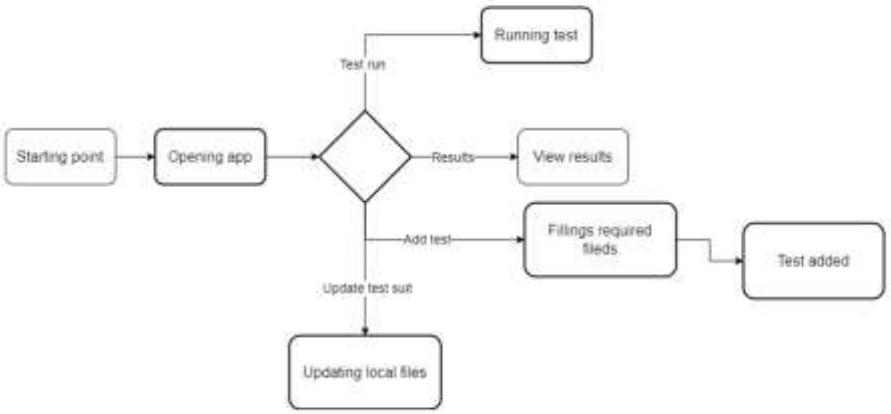
6

Use case діаграма додатку



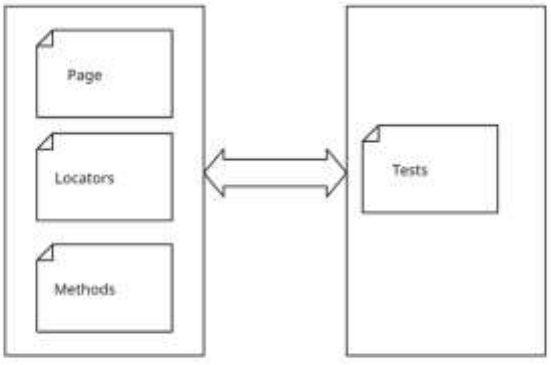
7

User flow додатку



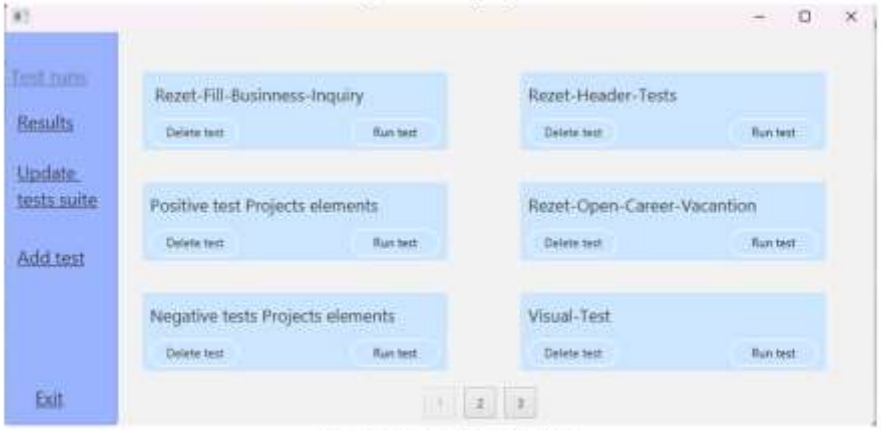
8

Структура патерну Page Object



9

Екранні форми



Користувацький інтерфейс

10

Екранні форми

Згенерований звіт з успішно пройденими тестами

11

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Волошин В.В. СУЧАСНІ ЗАСОБИ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ В ІКТ/Негоденко О.В, Волошин В.В// Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ». Збірник тез. 20.04.2023, ДУТ, м. Київ — К.: ДУТ, 2023. — С. 61-62.

Волошин В.В. СУЧАСНІ ЗАСОБИ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ В IOT / Негоденко О.В., Волошин В.В. // IV Науково-технічна конференція «Сучасний стан та перспективи розвитку IoT». Збірник тез. 07.04.2023, ДУТ, м. Київ — К.: ДУТ, 2023. — С. 40-42.

13

ВИСНОВКИ

1. Проведено аналіз архітектури існуючих інструментів автоматизації тестування. Було проведено дослідження різних інструментів тестування, враховуючи можливості кожного з них, вимоги проекту. Вибрано такі інструменти, як Cypress, Selenium, Mocha та інші.
2. Порівняно інструменти автоматизованого тестування та виявлено переваги та недоліки кожного з них.
3. Розроблено програмний продукт, що забезпечує автоматизоване тестування сайту компанії за допомогою Cypress. Створено різноманітні тестові скрипти, які перевіряли функціонал сайту з різних сторінок та різними способами.
4. Під час тестування програмного забезпечення перевірено роботу всіх функцій сайту та виявлено деякі помилки, які були виправлені. Крім того, проведено тестування на різних конфігураціях та середовищах, щоб забезпечити максимальну надійність та стабільність роботи програмного продукту.