ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ Кафедра інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської кваліфікаційної роботи

на ступінь вищої освіти бакалавр

на тему: **«Розробка програмного забезпечення з автоматизованого тестування інтернет-магазину мовою C#»**

Виконав: студент 4 курсу, групи ПД– 44 спеціальності

121 Інженерія програмного забезпечення

(шифр і назва спеціальності)

Дзененко В.О.

(прізвище та ініціали)

Керівник ст.в. Гаманюк І.М.

(прізвище та ініціали)

Рецензент_____

(прізвище та ініціали)

Нормоконтроль _____

(прізвище та ініціали)

Київ – 2022

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.
"____" _____2022 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТА

ДЗЕНЕНКО ВЛАДИСЛАВУ ОЛЕГОВИЧУ
(прізвище, ім'я, по батькові)

1. Тема роботи: «Розробка програмного забезпечення з автоматизованого тестування інтернет-магазину мовою C#»

Керівник роботи: Гаманюк І.М., ст.в.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом вищого навчального закладу від
2. Строк подання студентом роботи___
3. Вхідні дані до роботи
Методи тестування;
Науково-технічна література з питань, пов'язаних з тестуванням комп'ютерних систем та програм

4. Зміст розрахунково-пояснювальної записки(перелік питань, які потрібно розробити).
4.1 Автоматизований тест.
4.2 Використання та тестування системи.
4.3 Опис використаних технологій.

5. Перелік демонстраційного матеріалу (назва основних слайдів)
5.1 Об'єкт, предмет та мета
5.2 Постановка задачі

5.3 Порівняння аналогів інструментів тестування
5.4 Програми та технічні засоби
5.5 Архітектура веб-сторінок
5.6 Архітектура автоматизованого тесту
6. Дата видачі завдання «11» квітня 2022 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	02.04-07.04	Виконано
2	Вимоги до системи	08.04-10.04	Виконано
3	Створення та навчання моделі для вилучення полів	11.04-18.04	Виконано
4	Створення та навчання моделі для вилучення таблиць	19.04-28.04	Виконано
5	Концепція та архітектура програмного забезпечення	29.04-04.04	Виконано
6	Вступ, висновки, реферат	05.05-12.05	Виконано
7	Розробка обов‘язкових демонстраційних матеріалів	13.05-15.05	Виконано
8	Попередній захист роботи	28.05- 03.06	Виконано
9	Здача роботи	05.06.2022	

Студент ____ _____ Дзененко В.О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Гаманюк І.М._____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи 51 с., 25 рис., 8 джерел
Ключові слова: Selenium, C#, Website, DOM.

Об'єкт дослідження – тестування застосунку.

Предмет дослідження – програмне забезпечення тестування застосунку.

Мета роботи – підвищення швидкості процесу тестування.

Методи дослідження – автоматизація процесів тестування.

У роботі проведено аналіз існуючих видів тестування та моделей:
каскадна, спіральна і т.д.

Можливою проблемою інтернет-магазина може бути процес оформлення замовлення та передачі його в БД з подальшим опрацюванням.

Особливістю є можливість позбавлення тестувальника роботи над ручним тестуванням, за рахунок того що машина виконує тест набагато швидше та ефективніше. В такому випадку у тестувальника є можливість працювати на інших напрямках де він потребуєтьс поки тест автоматично виконується

В якості середовища розробки було взято IntelliJ IDEA та бібліотеку Maven для взаємодії з ним.

Отже, створено та описано роботу тестових сценарії для важливих фрагментів роботи веб-ресурсів.

Галузь використання – тестування.

Зміст

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

C#– мова програмування C#

MVC – Model View Controller

MVP – Model View Pattern

MVVM – Model View ViewModel

UML – United Modeling Language

HTML – Hyper Text Markup Language

DOM – Document Object Model

HTTP – Hypertext Transfer Protocol

ВСТУП

Інтернет-магазини дуже популярні в наш час, нині будь-яка людина з доступом до інтернету може зайти на сайт-магазин за будь-яким товаром. Тому стає важливим питання правильної навігації користувача по сайту та вибору замовлення. Особливу увагу потрібно приділити якраз вибору та замовленню товара/продукції, так як цей етап вважається вразливим для такого типу сайтів.

Додати, наприклад, футболку до кошика досить просто, з цим етапом проблем не буде. Але щоб оформити замовлення потрібно передати ці дані на склад, або базу даних. Де воно буде опрацьовуватися. Важливо щоб все було оформлено правильно, бо у випадку помилки користувач втрачає «товар», а сайт – дохід. Цей етап важливий, тому що він опрацьовується як зі сторони клієнта(покупця), так і зі сторони адміністратора. Можна також автоматизувати тестування процесу створення нових пунктків в каталозі товарів, так як він теж являється досить частим у використанні модулем. Важливо також забезпечити правильне опрацювання купівлі товару в інтернет-магазині через банківські системи і т.д., так як до питання грошей потрібно поставитись з відповідальністю.

Мета дослідження – програмне забезпечення тестування застосунку

Об’єкт дослідження – тестування застосунку

Предмет дослідження – підвищення швидкості процесу тестування

1 ОСНОВНІ ПОЛОЖЕННЯ

1.1 Веб-додатки та інтернет-магазин

Інтернет-магазин це – електронний ресурс, сайт з деяким каталогом, за допомогою якого відбувається продаж товарів користувачеві, враховуючи доставку. При цьому розміщення інформації, замовлення товару і угода відбуваються відразу на сайті магазину. Основна відмінність інтернет-магазину від звичайного магазину – полягає у відсутності потреби фізичних засобів. Якщо в звичайному магазині потрібен торговий зал, вітрини, цінники, продавці, касири, консультанти і охоронці, то у його мережевого конкуренту вся ця інфраструктура реалізована програмно.

У професійному інтернет-магазині обов’язково повинні бути визначені функціональні можливості та блоки для забезпечення зручної роботи сервісу з потенційним клієнтом. Для інформування користувача існують такі можливості: перелік товарів та послуг, прайс-листи, контактна інформація (адреса, телефон, факс, e-mail), пошук по сайту, умови співробітництва, методи оплати та доставки. Також важливими елементами інтернет-магазину, з точки зору роботи з ним, є наявність реєстрації та власного кабінету користувача, можливість додавати товари в «Кошик» (для подальшого замовлення бажаного товару), сторінка порівняння товарів, онлайн консультант та інші.

Незважаючи на схожість з роботою простого магазину, сьогодні інтернет-магазини стають одними з найбільш перспективних способів ведення і розвитку великого і малого бізнесу при зниженні витрат і збільшенні прибутку. І це не дивно, адже будь-який інтернет-магазин має ряд переваг перед звичайним магазином, а саме:

1) вартість і створення інтернет-магазину набагато дешевше і швидше, ніж самого простого маленького магазинчика;
2) широкий асортимент і часте оновлення товарів;
3) працює 24 години на добу і 365 днів на рік, без перерви, без вихідних (за винятком збоїв на сервері);
4) доступ до товарів має кожен відвідувач з будь-якої точки світу
5) низькі ціни, адже зазвичай інтернет-магазини працюють безпосередньо з постачальниками.

Про недоліки, то він тільки один. Товар неможливо побачити на живо, уважно розглянути, приміряти і т.д., але завжди можна повернути його протягом 14 днів, або оформити платіж після ознайомлення з ним.

Зовні схема роботи в усіх інтернет-магазинів приблизно однакова: покупець заходить на сайт, обирає товар і оформляє замовлення, після підтвердження замовлення відправляється до служби доставки, клієнт отримує товар і віддає за нього гроші (якщо вони не були перераховані раніше). Після чого починається постпродажна робота з покупцем (email-маркетинг, реклама тощо). Але ось функціонування цієї схеми може бути забезпечене по-різному.

Нижче приведений приклад функціональності інтернет-магазину:

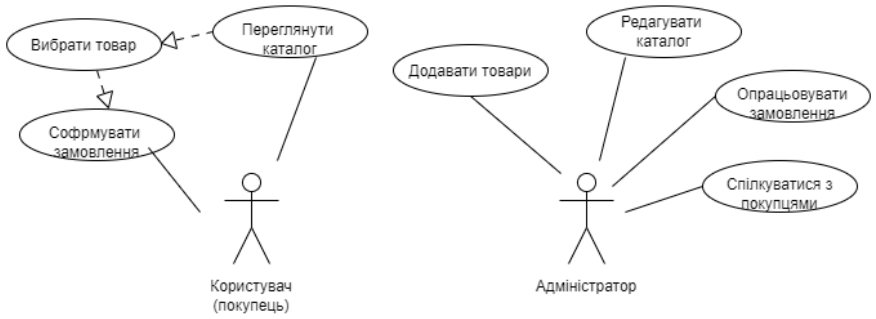


Рисунок 1.1 Функціональність інтернет-магазину

Веб-додаток є веб-сайтом, на якому розміщені сторінки з частково чи повністю несформованим вмістом. Остаточний вміст формується лише після того, як відвідувач сайту запросить сторінку із веб-сервера. У зв'язку з тим, що остаточне вміст сторінки залежить від запиту, створеного на основі дій відвідувача. Звідси можна зробити висновок, що веб-додаток передбачає щільна взаємодія з користувачем, отримання від нього «бізнес даних», їх складну обробку та зберігання, можливо навіть без надання результату користувачеві. Це означає, що відвідувач може взаємодіяти з матеріалом, функціями: натискати кнопки, заповнювати форми, вимагати прайс, робити покупки. Практично будь-який Інтернет-ресурс входить до веб-додатків: пошукові системи, відео сервіси, соціальні мережі, а також будь-які веб-сайти з функціями автентифікації користувача, купівлі товарів та послуг, замовлень, бронювання квитків тощо.

Технічно веб-програма має клієнт-серверну архітектуру.

Клієнтом є браузер, а сервером є веб-сервер. Зв'язок відбувається з допомогою методів запитів. Найпоширеніші методами запитів є запити типу GET та POST. Метод GET, як і метод POST може використовуватися і для надсилання даних, і для отримання інформації з веб-сервера. Метод GET надсилає серверу всю зібрану інформацію форми, як частина URL. Метод

POST передає дані таким чином, що користувач уже не бачить дані, що передаються серверу. Зазвичай через метод GET передають текст, короткі запити, а метод POST передають великий обсяг даних. Візуалізація методів представлена рисунку 1.2.

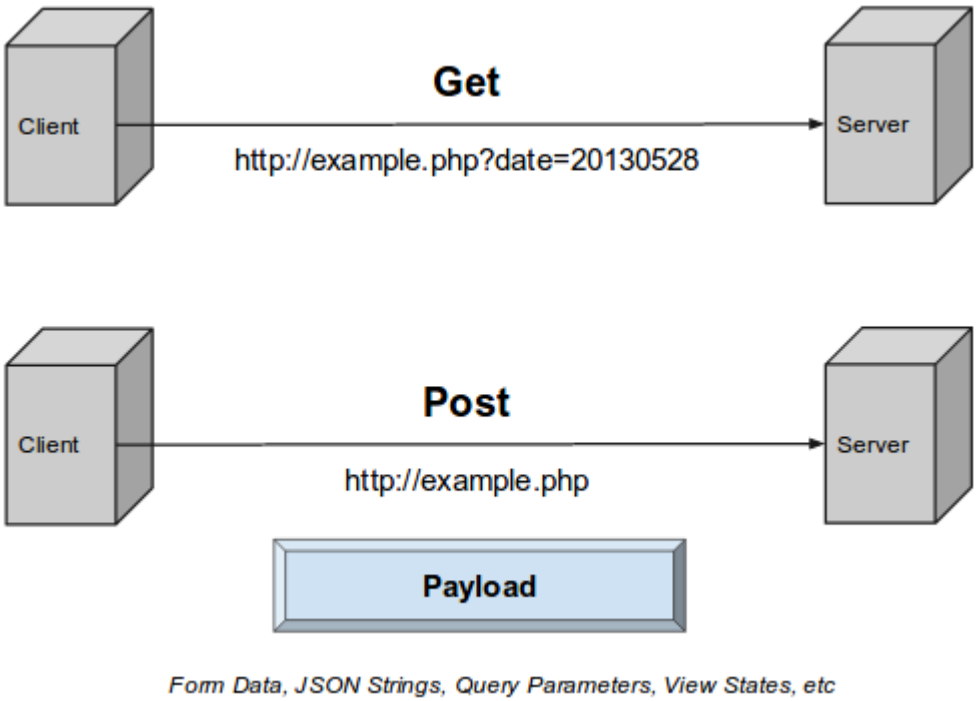


Рисунок 1.2 – Методи Get і Post

Спочатку веб-додаток складається зі сторінок з частково або повністю невизначеним вмістом. Підсумковий вміст веб-сторінок сформується тоді, коли конкретний користувач надішле запит на сервер.

Розглянемо структуру веб-сторінки. Веб-сторінка складається з трьох частин:

1) Інформація про версію HTML. Інформація про версію HTML представляється у вигляді першого рядка у вихідному коді веб-сторінки;
2) Шапка веб-сторінки, де міститься технічна інформація про веб-сторінку, наприклад, назву, ключові слова, метадані. Являє собою набір елементів,

які не входять до графічного представлення веб-сторінки. Шапка веб-сторінки обмежується елементом (тегом) <head>…</head>;

3) Тіло веб-сторінки, яке містить графічне та інформаційне подання веб-сторінки. Тіло веб-сторінки обмежується елементом (тегом) <body>…</body>.

Така структура є стандартною для всіх веб-сторінок. Сторінки, які ми бачимо у браузері, можуть бути статичними та динамічними. Статична веб-сторінка відображається для всіх відвідувачів однаково. Алгоритм виглядає так:

1) Користувач вводить в адресному рядку браузера запит або адресу сторінки;
2) Браузер надсилає його на веб-сервер;
3) Запит аналізується на сервері, визначається, що жодних особливих ознак та інструкцій для користувача немає;
4) Сервер надсилає веб-сторінку браузеру без зміни будь-яких даних у ній. Наприклад, це матеріал новин, загальна стандартна інформація.

У випадку з динамічними сторінками:

1) Користувач вводить в адресному рядку браузера запит або адресу сторінки;
2) Браузер надіслав запит до веб-сервера з інформацією, що у цього користувача є набір ознак, за наявності яких для нього потрібно показувати певну інформацію;
3) Веб-сервер пересилає набір ознак сервер додатків, де застосовуються правила та інструкції для додавання особливих змінних. Наприклад, якщо користувач авторизований у системі, то йому може здатися сторінка з ПІБ або іншою релевантною для користувача інформацією;

4) Сервер забирає готову веб-сторінку, відправляє браузеру, який показує її відвідувачу, який створив запит. Приклад візуалізації статичних та динамічних веб-сторінок представлена рисунку 1.3

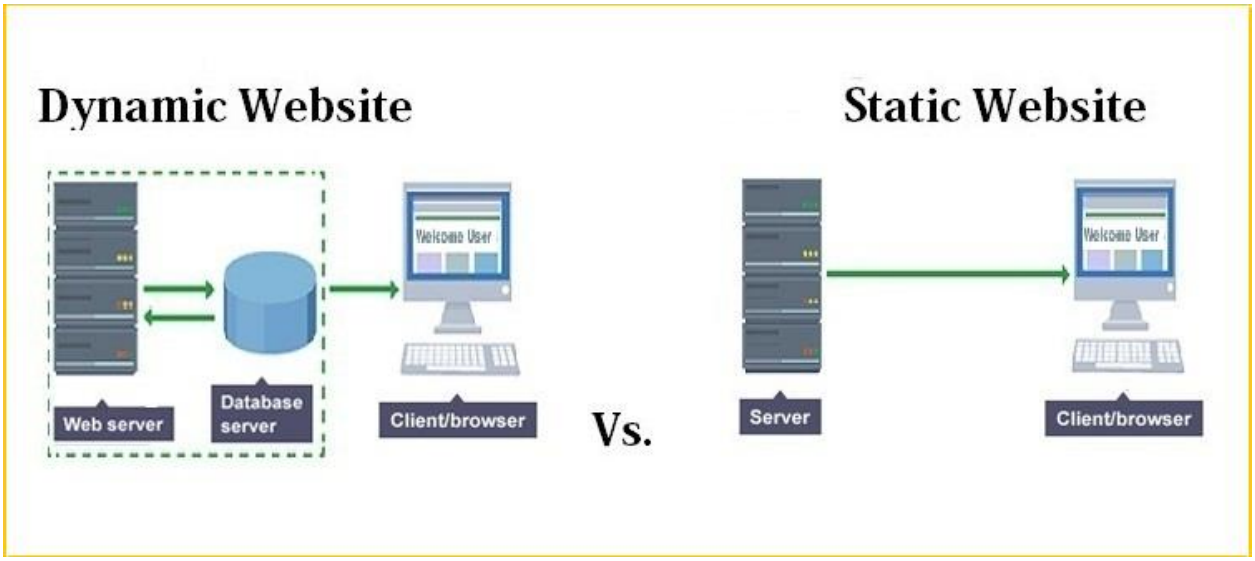


Рисунок 1.3 – Динамічні та статичні сайти

Технічно є можливість динамічно оновлювати контент або окремі блоки на сторінці без перезавантаження сторінки. Для цього використовується AJAX та jQuery. AJAX – інструмент для побудови веб-додатків, що обмінюються даними з сервером у фоновому режимі.

При цьому користувач отримує програму з динамічним оновленням контенту, без перезавантаження всієї сторінки. jQuery – JavaScript-фреймворк, бібліотека, що дозволяє зручніше використовувати деякі можливість JavaScript, такі як: створення візуальних ефектів, обробка подій, робота з DOM та підтримка AJAX. Основний принцип роботи динамічного оновлення блоків на сторінці виглядає наступним чином:

1) Створюється функція та об'єкт для браузерів. Цей об'єкт потрібен для передачі даних на сервер та отримання від нього відповіді у фоновому режимі, без перезавантаження сторінки;
2) У тілі документа створюються два контейнери. У перший контейнер завантажується безпосередньо контент, а другий контейнер буде містити контент, який служить заставкою і з'являється на час завантаження основного необхідного контенту;
3) Створюється додаткова функція, яка виводить контент у перший контейнер. Функція робить це за допомогою раннього створеного об'єкта. Визначеним методом здійснюється опис передачі на сервер. Як параметри вказується тип запиту (наприклад, GET) та рядок, що передається серверу (URL сторінки, що завантажується);
4) Для отримання контенту потрібно його дочекатися. Як тільки його чекаємо, то відразу змінюється тіло контейнера;
5) Після відкриття запиту запит надсилається на сервер.

Основним інструментом роботи з динамічними змінами на веб-сторінка є документно-об'єктна модель (DOM). Вонавикористовується для документів XML/HTML. Згідно з DOM-моделлю, документ є ієрархією, деревом. Кожен HTML-тег утворює вузол дерева, тип якого елемент. Вкладені до нього теги стають дочірніми вузлами. Щоб уявити текстову інформацію створюються вузли, типом яких є текст.

Усього розрізняють 12 типів вузлів, але практично працюють з чотирма з них:

1) Документ – точка входу до DOM;
2) Елементи – основні будівельні блоки;
3) Текстові вузли – містять, власне, текст;
4) Коментарі - іноді в них можна включити інформацію, яка не буде показано, але доступне з JavaScript.

Отже, DOM є поданням документа у вигляді дерева об'єктів, яке доступне для редагування через JavaScript. DOM потрібний, щоб керувати сторінкою, наприклад, читати інформацію з HTML, змінювати та створювати елементи. Ілюстрація DOM-дерева представлена рисунку 1.4.

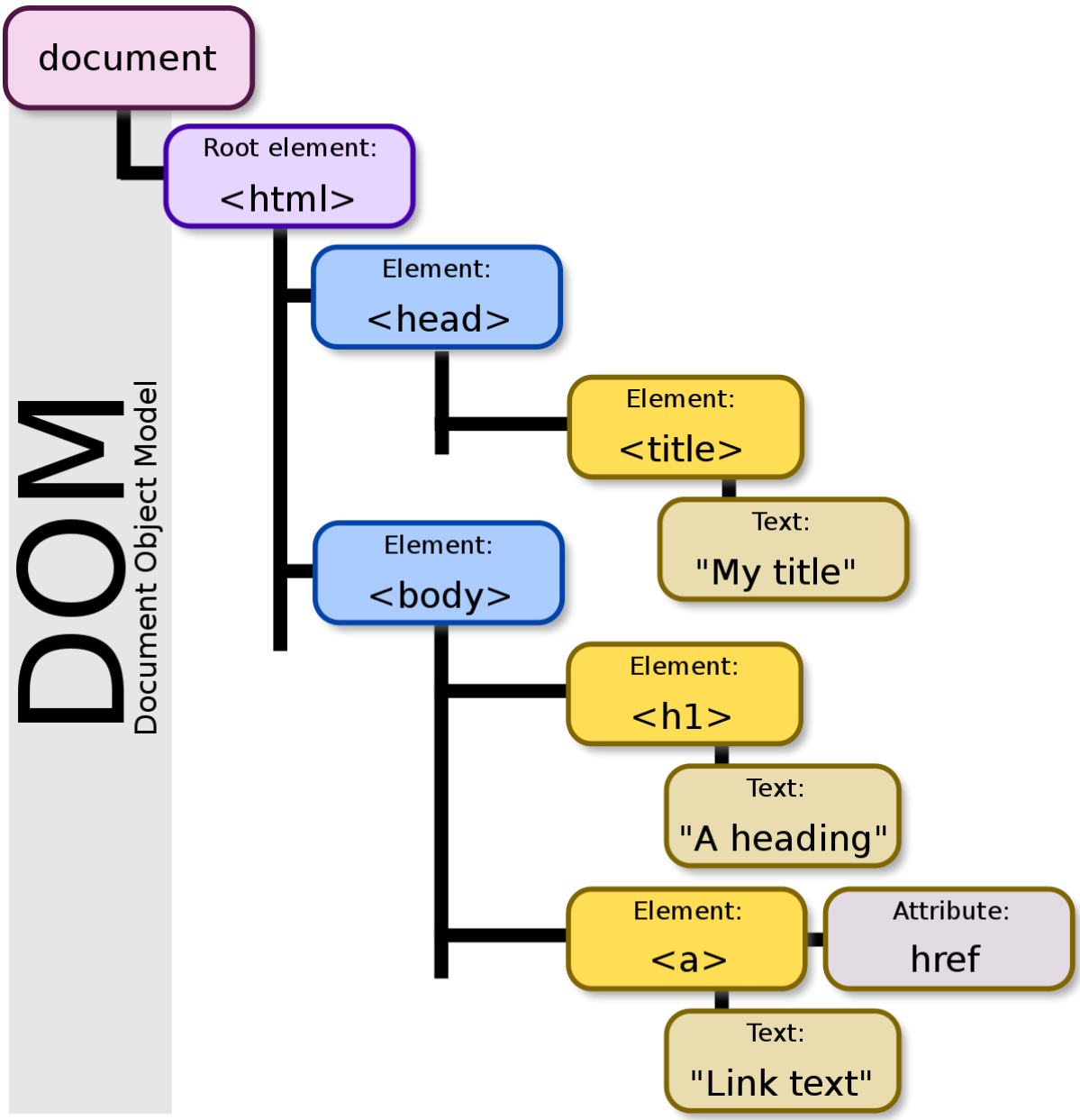


Рисунок 1.4 – DOM архітектура

Як правило, у реальних завданнях потрібно маніпулювати наборами елементів (або одним), що знаходяться десь глибоко в DOM-дереві, причому, найчастіше, ці елементи розкидані на його різних частинах. Наприклад,

потрібно зазначити на веб-сторінці список файлів на видалення та виконати цю дію. У такій ситуації ручний прохід по дереву виявиться вкрай стомлюючим заняттям. У цьому випадку пошук елемента DOM-дереву можна здійснити за допомогою спеціалізованих пошукових методів:

1) Ідентифікатор елемента By.id. Якщо елемент призначений спеціальний атрибут id, то можна отримати його прямо по змінної з ім'ям значення id. За стандартом значення id має бути унікально, тобто в документі може бути лише один елемент із даними id. І саме його буде повернено. Якщо в документ має кілька елементів з унікальним id, то поведінка буде невизначено. В цьому випадку браузер поверне елемент випадковим чином. Тому при розробці веб-додатків намагаються дотримуватися правила унікальності id.
2) На ім'я тега елемента By.tagName. Цей спосіб дозволяє отримати всі елементи з певним тегом і серед них шукати потрібний. Регістр тега не має значення.
3) Назву елемента By.Name. Виклик цього методу дозволяє отримати всі елементи з атрибутом name. Використовується цей метод дуже рідко, т.к. Назва часто в DOM-дереві неунікальне.
4) На ім'я класу By.className. Метод повертає колекцію елементів із класом className. Знаходить елемент у тому випадку, якщо має кілька класів, а шуканий – одне із них.
5) За селектором таблиць стилів By.cssSelector. CSS (Cascading Style Sheets – каскадні таблиці стилів) є формальною мовою описи зовнішнього вигляду документа, написаного з використанням мови розмітки. Метод повертає елементи, що задовольняють CSS-селектору. Це один із найчастіше використовуваних та корисних методів під час роботи з DOM. Структура CSS-селектор враховує ієрархію об'єктів у DOM-дереві, а також критерії на основі тегів та атрибутів. Напрями руху по DOM-дереву лише у глибину.

6) За допомогою мови запитів до елементів XML (eXtensible Markup Language – мова розмітки, що розширюється) By.xpath. Xpath (XML Path Language) – мова запитів до елементів XML документа, реалізує навігацію по DOM-дереву. Метод є популярним, як пошук за CSS-селектором. Xpath також враховує ієрархію об'єктів у DOM-дереві, а також критерії на основі тегів та атрибутів. Однак, напрям руху по DOM-дереву, на відміну від CSS-селектора може бути в будь-якому напрямку.

1.2 Каркаси веб-застосунків

На сьогоднішній день, більшість веб-додатків мають у себе На основі деякий каркас. Цей каркас прийнято називати фреймворком.

Фреймворки – це програмні продукти, які спрощують виробництво та підтримку технічно складних чи навантажених проектів [9]. Фреймворк, як правило, містить у собі лише самі базові програмні модулі, а всі інші для проекту компоненти реалізуються розробником на їхню основу. В результаті чого досягається висока швидкість розробки ПЗ та велика продуктивність та надійність рішень. Фреймворки можуть підходити і для створення різних сайтів, бізнес-додатківта веб-сервісів.

Фреймворк відрізняється від бібліотеки тим, що бібліотека може бути використана у програмному продукті як набір підсистем близької функціональності, не впливаючи на архітектуру основного програмного продукту, і навіть не накладаючи неї ніяких обмежень. Фреймворк ставить базове побудови архітектури програми, формуючи на початковому етапі розробки за умовчанням поведінка, формуючи каркас програми, який необхідно буде розширювати, а також змінювати відповідно до зазначених вимог. Додатково у фреймворку може утримуватися допоміжні програми, різні бібліотеки, мова сценаріїв та інше ПЗ, яке допомагає полегшити розробку та об'єднання різнихкомпонентів програмного проекту

Головна перевага при використанні фреймворків – стандартизована структура організації компонентів програми. Створення цієї структури розробки на фреймворках дуже спрощується. По суті, фреймворк - це безліч конкретних та абстрактних класів, а також опис способів їх взаємодії.

Конкретні класи зазвичай реалізують взаємні зв'язки, а абстрактні – є розширення, в яких фреймворки можуть бути адаптовані або використані. Для забезпечення розширення можливостей зазвичай використовуються техніки об'єктно-орієнтованого програмування (наприклад, частини програми можуть успадковуватися від базових класів фреймворку).

Виділяють деякі переваги розробки веб-додатків з використанням фреймворків:

1) Розробка на фреймворку, на відміну від так званих «самописних» рішень, дозволяє домогтися простоти супроводжуваності проекту;
2) Відносна проста реалізація будь-яких бізнес-процесів, а не тільки тих, що були спочатку закладені в систему. Проекти на базі каркасів можуть бути легко масштабовані та модернізовані;
3) Системи, реалізовані при промощі фреймворків, часто, працюють швидше, а також витримують велике навантаження, порівняння з самописними системами. Саме тому багато хто популярні веб-застосунки мають у основі фреймворки. За рівню безпеки дані рішення також значно перевершують самописні системи.

Більшість фреймворків веб-додатків реалізує шаблон проектування Model-View-Controller (MVC) Модель (Model) надає дані та реагує на команди контролера, змінюючи своє стан. Подання (View) відповідає за відображення даних моделі користувача, реагуючи на зміни моделі. Контролер (Controller) інтерпретує дії користувача, сповіщаючи модель про необхідність змін. Діаграма шаблону MVC представлена рисунку 1.5.

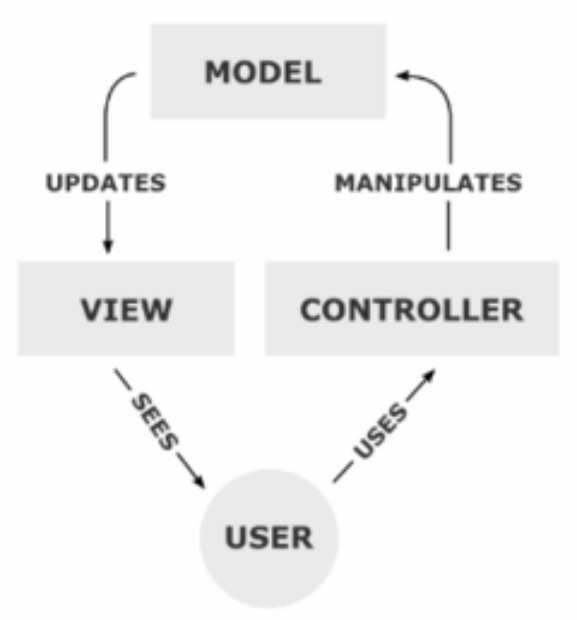


Рисунок 1.5 – Шаблон MVC

Однак, також можуть використовуватися інші шаблони, наприклад, Model-View-Presenter (MVP) або Model-View-ViewModel (MVVM). Вони є похідними шаблонами MVC.

Елемент Presenter у шаблоні Model-View-Presenter бере на себе функціональність посередника (аналогічно Контролеру в MVC) та відповідає за керування подіями інтерфейсу користувача (наприклад, використання миші) так само, як в інших шаблонах зазвичай відповідає уявлення. Діаграма шаблону MVP представлена рисунку 1.6.

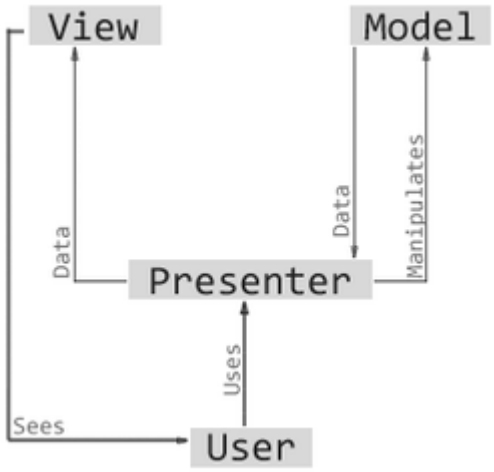


Рисунок 1.6 – Шаблон MVP

Модель Подання (ViewModel) у шаблоні Model-View-ViewModel є, з одного боку, абстракцією Уявлення, а з іншого, надає обгортку даних із Моделі, які підлягають зв'язуванню. То є, вона містить Модель, яка перетворена на Подання, а також містить у собі інструкції, якими може користуватися Подання, щоб впливати на модель. Діаграма шаблону MVVM представлена на рисунку 1.7.

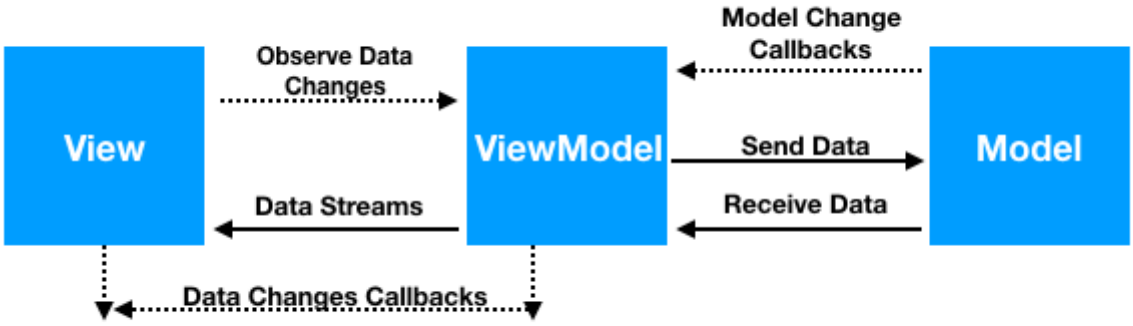


Рисунок 1.7 – Шаблон MVVM

У шаблонах проектування MVC/MVP зміни в користувальницькому інтерфейс не впливають безпосередньо на модель, а попередньо йдуть через Controller чи Presenter. У деяких технологіях, наприклад, Silverlight є концепція «зв'язування даних», що дозволяє пов'язувати дані з візуальними елементами з обох боків. Тому, застосування моделі MVC стає незручним, тому що прив'язка даних до подання безпосередньо не укладається у концепцію MVC/MVP.

1.3 Тестування програмного забезпечення

Тестування програмного забезпечення (Software Testing) – перевірка відповідності між реальною та очікуваною поведінкою програми, що здійснюється на кінцевому наборі тестів, обраному певним чином. У більш широкому сенсі, тестування є однією з технік контролю якості, яка включає в себе активності по планування робіт, проектування тестів, виконання тестування та аналізу одержаних результатів. Розглянемо види тестування ПЗ. Всі види тестування програмного забезпечення, залежно від переслідуваних цілей, можна умовно поділити на такі групи:

1) Функціональні;
2) Нефункціональні;
3) Пов'язані зі змінами.

Розглянемо кожен окремий вид тестування, його призначення та використання під час тестування ПЗ.

Функціональні тести ґрунтуються на функціях та взаємодії з іншими системами. Тести також можуть бути представлені на всіх рівнях тестування: компонентному або модульному, інтеграційному, системному та приймальному. Функціональні види тестування розглядають зовнішнє

поведінка системи. Розглянемо одні з найпоширеніших видів функціональних тестів:

1) Тестування взаємодії;
2) Тестування безпеки;
3) Функціональне тестування.

Нефункціональні тести, необхідні для визначення характеристик програмного забезпечення, які можуть бути виміряні різними величинами. У широкому сенсі, нефункціональне

Тестування - це тестування того, як працює система. Основні види дисфункційних тестів:

1) Тестування продуктивності:
   a) тестування навантаження;
   b) стресове тестування;
   c) тестування стабільності чи надійності;
   d) об'ємне тестування.
2) Тестування установки;
3) Тестування зручності використання;
4) Тестування на відмову та відновлення;
5) Конфігураційне тестування.

Після проведення необхідних змін, таких як виправлення дефекту, ПЗ має бути перетестовано для підтвердження, що проблема була справді виправлена. Нижче перераховані види тестування, які необхідно проводити після встановлення програмного забезпечення, для підтвердження працездатності програми або правильності здійсненого виправлення дефекту:

1) Димове тестування;
2) Регресійне тестування;
3) Тестування збирання;

4) Санітарне тестування (перевірка узгодженості).

Тепер розберемося в основних поняттях та техніках тестування веб-додатків.

Верифікація – це оцінка системи чи її компонентів із метою визначення результатів поточного етапу розробки умов, які були сформовані на початку цього етапу. Наприклад, чи виконуються цілі, не чи зриваються терміни, завдання з розробки проекту, які були визначені на початку поточної фази.

Валідація – це визначення відповідності, що розробляється програмного забезпечення очікуванням та потребам бізнес-замовника, вимог до системи.

План тестування – це документ, який описує повний обсяг робіт з тестування, починаючи з опису об'єкта, стратегії, розкладу, критеріїв початку та закінчення тестування, до необхідного у процесі роботи обладнання, спеціальних знань, а також оцінки ризиків з варіантами їх вирішення.

Хороший план тестування має містити відповіді на такі питання:

1) Що потрібно тестувати?

Відповідь на це питання має містити опис об'єкта тестування: додатки, системи, обладнання.

2) Що саме необхідно тестувати?

Відповідь на це питання має містити опис тестованої системи та її компоненти окремо, список функцій.

3) Як потрібно тестувати?

Відповідь на це питання має описувати стратегію тестування, наприклад, види тестування та їх застосування до об'єкт тестування.

4) Коли потрібно тестувати?

Відповідь на це питання має містити послідовність проведення робіт: підготовка до тестування, безпосередньо тестування та аналіз результатів за запланованими етапами розробки.

5) Які критерії потрібні, щоб розпочати тестування?
   a) готовність тестового тестового стенду;
   b) закінчення етапу розробки необхідного функціоналу;
   c) повнота документації.
   d) Які критерії необхідні для закінчення тестування?

Відповідь на це питання має містити результати тестування, що задовольняють критеріям якості продукту:

1) вимоги до кількості відкритих дефектів виконані;
2) витримка певного періоду без зміни вихідного

коду програми;

1) витримка певного періоду без відкриття нових багів.

Відповівши у своєму плані тестування на ці питання, можна бути впевненим, що вже є чудова чернетка документа з планування тестування. Для того, щоб чернетка перетворилася на серйозну документ, необхідно доповнити його такими пунктами:

2) Оточення тестованої системи (опис програмно-апаратних засобів);
3) Необхідне для тестування обладнання та програмні засоби (тестовий стенд та його конфігурація, програми для автоматизованого тестування тощо);
4) Ризики та шляхи їх вирішення.

Розглянемо поняття тестового випадку.

Тестовий випадок (тест-кейс) -це документ, який описує сукупність кроків, конкретних умов та параметрів, необхідних для перевірки реалізації функції, що тестується чи її частини. Під тест-кейсом розуміється структура виду:

Дія	Очікуваний результат	Результат тесту

1.3.1 Застосування автоматизованого тестування

Економічна доцільність

З технічної точки зору, будь-які завдання тестування можуть бути автоматизовані. Або, принаймні, більшість з них. Питання в тому, яких витрат це вимагатиме. Деякі види тестування простіше, дешевше і швидше виконати вручну. Наприклад, оцінити зручність графічного користувацького інтерфейсу: людина може на око оцінити розташування елементів керування, їх компонування, кольорову гамму, логічність інтерфейсу.

Так, більша частина таких перевірок може бути реалізована програмно. Ось тільки час, витрачений на створення відповідного алгоритму, його валідацію, імплементацію, виконання та аналіз отриманих результатів, може в багато разів перевищувати час і вартість виконання такої перевірки вручну. Щоб уникнути цього, на етапі підготовки автоматизації тестування підраховують економічну доцільність автоматизації (Return On Investments, ROI).

Регресійне та димове тестування

Однак для багатьох видів тестування автоматизація є економічно обґрунтованою. В першу чергу, для таких трудомістких і відносно довготривалих видів тестування, як регресійне та димове тестування. Їх особливість — в багаторазовому повторі великої кількості тестів в однакових умовах. Тому їх легко налагодити і потім виконувати раз за разом.

Unit-тестування

Існують типи тестування, які в принципі неможливо виконати в ручному режимі. Наприклад, модульне (unit) тестування: перевірити мільйони операторів, переходів та їх ланцюжки, комбінації на всіх можливих наборах даних вручну неможливо. Або тестування продуктивності: яким чином вручну імітувати одночасне відвідування сайту тисячами обо навіть десятками тисяч користувачів? Очевидно, що без спеціального ПЗ не обійтися.

Accessibility-тестування

Існують також специфічні види тестування, які дають додаткову інформацію про якість додатку, необхідність в якій обумовлена предметною галуззю або умовами застосування. Наприклад, тестування доступності (accessibility), яке характеризує можливість використання додатку користувачами з особливими потребами.

Вимоги щодо доступності публічних комерційних і державних сервісів для таких груп користувачів наразі регулюються законодавством більшості розвинених країн, і замовники наполягають на тому, щоб продукти, що розробляються, обов’язково перевірялися на доступність. Ці перевірки технічно нескладні, проте численні та рутинні. Ручні перевірки можуть тривати кілька днів, тоді як їх автоматизація інструментами, наявними у вільному доступі, доволі проста, а виконання основної частини перевірок займатиме не більше кількох хвилин.

Інтеграція автоматизації в розробку ПЗ

Звичайно, не можна залишити осторонь питання інтеграції автоматизованих видів тестування в сучасні процеси розробки ПЗ (Software Development Life Cycle, SDLC). Сучасний стандарт передбачає застосування концепції Continuous Integration and Deployment (CI/CD). Всі сучасні

інструменти автоматизації тестування розраховані на швидку і максимально просту, «безшовну», інтеграцію з інструментами CI/CD — Jenkins, Bambo, GitHub, хмарними сервісами на зразок AWS або Azure. Таким чином, процедури автоматичного тестування і розгортання нової збірки можна запустити натисканням однієї кнопки.

Види автоматизованого тестування поділяються на:

1) Unit-тестування (або модульне):
   a) Що тестується: Правильність роботи класів та методів;
   b) Хто тестує: Розробник, який написав код;
   c) Навіщо: Щоб відловити найочевидніші помилки на стадії написання коду.
2) Тестування API:
   a) Що тестується: REST чи SOAP сервіси;
   b) Хто тестує: Розробники сервісів або фахівці з автоматизації тестування;
   c) Для виявлення помилок на стадії тестування компонентів.
3) UI-тестування:
   a) Що тестується: функціонал графічного інтерфейсу;
   b) Хто тестує: спеціалісти з автоматизації тестування;
   c) Навіщо: для полегшення роботи ручних (функціональних) тестувальників та найбільш швидкого виявлення помилок на стадії приймального тестування.

Юніт-тести – це тестування одного модуля коду (зазвичай це одна функція або один клас у разі ООП-коду в ізольованому оточенні. Це означає, що якщо код використовує якісь сторонні класи, то замість них підставляються класи заглушки (моки та стаби), код не повинен працювати з мережею (і зовнішніми серверами), файлами, базою даних (інакше ми тестуємо не саму функцію чи клас, а ще й диск, базу, тощо).

Стаби – це класи-заглушки, які замість виконання дії повертають якісь дані. Наприклад, стаб класу роботи з базою даних може замість реального звернення до бази даних повертати, що запит успішно виконано. А при спробі прочитати щось із неї повертаєзаздалегідь підготовлений масив із даними.

Моки – це класи-заглушки, які використовуються для перевірки, що певна функція була викликана. Зазвичай юніт-тест передає функції різні вхідні дані та перевіряє, що вона поверне очікуваний результат. Наприклад, якщо у нас є функція перевірки правильності номера телефону ми даємо їй заздалегідь підготовлені номери, і перевіряємо, що вона визначить їх правильно. Якщо у нас є функція розв'язання квадратного рівняння, ми перевіряємо, що вона повертає правильне коріння (для цього ми заздалегідь робимо список рівнянь із відповідями).

Юніт-тести добре тестують такий код, який містить якусь логіку. Якщо в коді мало логіки, а переважно містяться звернення до іншим класам, то юніт-тести написати може бути складно.

API – це набір функцій, які можна викликати, щоб отримати якісь дані. Наприклад, сервіс Яндекс.Карти має свій API геокодера. Надіславши до нього запит із географічною адресою, ти можеш отримати координати точки (і навпаки), а Центробанк має API, яке повертає офіційний курс валют у певний день. Якщо будь-яка програма має API, то можна тестувати її, посилаючи заздалегідь підготовлені запити і порівнюючи відповідь, що прийшла з очікуваним.

GUI – це графічний інтерфейс, тобто те, що користувач бачить на екрані. Це найскладніший для тестування вид. Якщо йдеться про перевірки роботи сайту, то ми повинні емулювати роботу браузера, який досить складно влаштований, аналізувати інформацію, що виводиться на сторінці. Але цей вид тестування дуже важливий, тому що він взаємодіє з додатком

так само, як і користувач. GUI тести ще називають End-to End (E2E) або приймальні (aceptance) тести.

Зробимо проміжний висновок про те, що потрібно автоматизувати:

1) Важкодоступні місця у системі (backend-процеси, логування файлів, запис у БД);
2) Часто використовувана функціональність, ризики від помилок, у яких досить високі. Автоматизувавши перевірку критичної функціональності, можна гарантувати швидке перебування помилок, отже, і швидке їх вирішення;
3) Рутинні операції, такі як перебори даних (форми з великим кількістю полів, що вводяться. Автоматизувати заповнення полів різними даними та їх перевірку після збереження);
4) Перевірка даних, які потребують точних математичних розрахунків;
5) Перевірка правильності пошуку.

При тестуванні не варто впадати у крайності. Наприклад, не можнаказати, що 100% веб-програми має бути покрито автотестами. Автоматизовані тести мають насамперед підвищувати якість коду, а також перевіряти роботу найбільш критичного функціоналу. Це вимагає досить багато часу на їх написання, налагодження, підтримку. Якщо ці витрати більше, ніж приноситься від них вигода, можливо, вони не потрібні.

Наприклад, якщо розробляється невеликий сайт, який потім не потрібно підтримувати, то найпростіше перевірити його вручну, ніж витрачати час написання автоматизовані скрипти.

З іншого боку, якщо велика команда працює над складним веб-додатком, то автотести необхідні, інакше більшу частину часу йтиме на пошук за допомогою ручного (функціонального) тестування та виправлення зламаного функціоналу. На жаль, не скрізь при розроблення складного

функціоналу впроваджено автоматизоване тестування. Десь програма перевіряється людьми. Не можна відкидати людський фактор. Люди можуть втомитися, можуть бути ліниві або неуважні, тоді як «машина» готова цілодобово виконувати одну й ту саму послідовність дій.

Важливо пам'ятати:

1) Автотести мають бути повторюваними. Не можна отримувати вихідні дані генератором випадкових чисел, тому що в цьому випадку ми не зможемо повторити тест;
2) Автотести повинні виконуватись у контрольованому оточенні;
3) Автотест не повинен використовувати той самий алгоритм, що і код, що перевіряється (функціонал), так як у цьому випадку в них може бути зроблена та сама помилка. Помилкові результати в цьому у разі співпадуть.
4) Необхідно тестувати позитивні та негативні сценарії. До наприклад, при тестуванні форми реєстрації треба перевірити не тільки як вона працює при введенні правильних даних, але і як вона працює з неправильними даними. За негативного сценарію має з'являтись повідомлення про помилку;
5) При виконанні автотестів слід відстежувати ті, що виникають помилки. Якщо у разі виникнення помилки просто виводиться повідомлення, яке автотест не обробляє, та програма продовжує виконуватися, це марний тест.
6) Автотести повинні бути легко запущені, в ідеалі однією командою. Якщо його запуску необхідно виконати багато дій, то людям буде лінь це робити. У компаніях зазвичай налаштовують CI сервер, який сам забирає оновлення з репозиторія, запускає тести та розсилає розробникам повідомлення при помилках.

Висновки до першого розділу

У цьому розділі було проведено аналіз різних видів тестування та їх переваг а також сутності веб-додатків та визначено основні поняття. Внаслідок чого були досліджені основи взаємодії клієнтської та серверної частин веб-додатків, структури веб-сторінок, варіанти пошуку веб-елементів, каркаси сучасних веб-додатків та огляд найзручнішого і найефективнішого вида тестування для цього типу веб-додатків.

2 РОЗРОБКА ТА АНАЛІЗ СИСТЕМИ ДЛЯ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ НА C#

2.1 Завдання автоматизованого тесту веб-ресурсу на C#

Тестування займає важливе місце в розробці програмного забезпечення. Саме воно дозволяє отримати якісний продукт і саме такий який буде зручний для використання. В тестуванні існує два види тестування: мануальне та автоматизоване. Зазвичай виконується мануальне тестування, але воно супроводжується нудною та монотонною роботою, що впливає на швидкість та концентрацію співробітника що працює над тестом. Саме в таких випадках використовують автоматизацію тестування. Достатньо лише автоматизувати процесс проходження тестового сценарію, і не витрачати на нього зайві ресурси. Він виконується швидко, та може бути пройдений велику кількість разів і одночасно якщо того потребує процес тестування. Але у автоматизованого тестування також є свої недоліки серед яких:

1) Автоматизація не дозволяє провести ефективне юзабіліті системи;
2) Складність написання автоматизованого тесту;
3) Підтотовка та тренування спеціалістів;
4) Зміна структури автоматизованого тесту при оновлені системи;

Як ми бачимо, обидва види тестування мають свої недоліки, але проблема вирішується комбінацією двох видів тестування. У випадку даної дипломної роботи буде вибрано тільки автоматизоване тестування, так як буде затронутий процес, що відтворюється велику кількість разів кожен день, та має важливе та критичне місце в системі.

Створення автоматизованого тесту дозволить вирішити такі задачі:

1) Збереження часу від витрат на мануальне тестування для задіяння в інших задачах;
2) Автоматизація часто повторюваного процесу що має безліч варіантів використання;
3) Тестування критично важливого процесу в описуваній системі;
4) Велика кількість виконуваних тестів за невеликий проміжок часу;

Мета роботи – підвищення швидкості процесу тестування.

2.2 Моделювання об’єкту проектування

2.2.1 Діаграма прецедентів системи

Щоб правильно створити тест потрібно знати процес який буде проходити користувач на сайті.

Проектування – один з важливих кроків при розробці програми, який дуже часто ігнорується розробниками-початківцями. Зазвичай вони намагаються втримати все у голові або, у кращому разі, записати деякі важливі відомості на аркуші паперу. Як результат, вони не мають чіткого плану подальших дій, і проект може бути відкладений у довгий ящик.

Зазвичай, при проектуванні розробники зображують систему графічно, оскільки людині легко розібратися в такому уявленні. Саме тому замість написання громіздких текстів про кожну нагоду майбутньої програми розробники будують різні діаграми для опису своїх систем. Це допомагає їм не забувати, що потрібно реалізувати у програмі, та швидко вводити в курс справи своїх колег. Для цього потрібно зрозуміти як працює система і зазвичай для цього використовують опис функціональності системи через варіанти використання (Use case або прецеденти). Варіанти використання –

це опис послідовності дій, які може здійснювати система у відповідь на дії та вплив користувачів або інших програмних систем. Побудова моделі варіантів використання дозволяє краще зрозуміти важливість функції відображених в системі для користувача.

Для процесу тестування, варіанти використання мають важливу роль, так як дозволяють оцінити точність реалізації вимог користувача. Також буде зрозуміло на яку функцію потрібно звернути найбільше уваги при тестуванні. Все це дозволить правильно та зручно налаштувати функції системи.

Діаграма варіантів використання складається з акторів, для яких система виробляє дію і власне дії Use Case, яке описує те, що актор хоче отримати від системи. Актор позначається чоловічка, а Use Case – овалом

Метою даного проекту є створення автоматизованого тесту про, важливий для сайту, процес оформлення замовлення в інтернет-магазині.

Користувачем даної системи виступає покупець та співробітник сайту. На рисунку 2.1 Показано основні дії в даній системі.

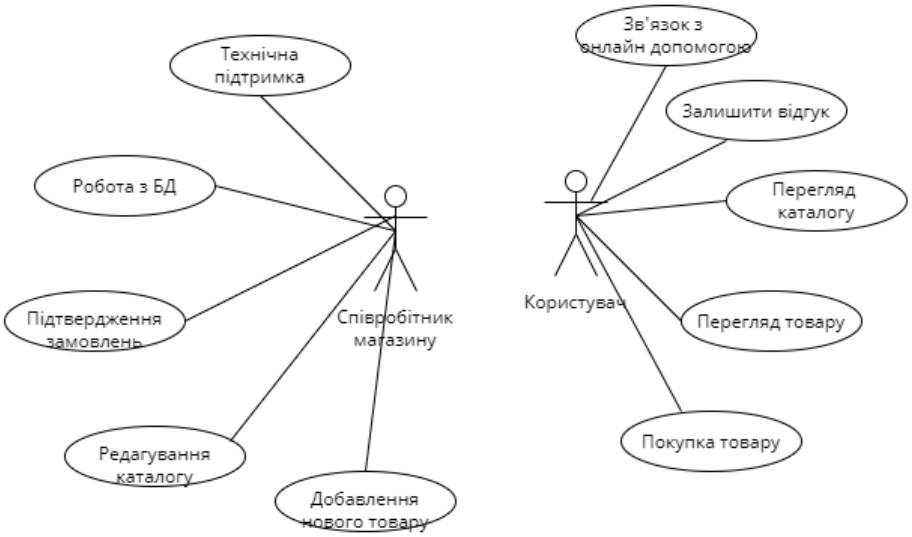


Рисунок 2.1 – Діаграма прецедентів

2.2.2 Діаграма діяльності

Діаграма діяльності використовується для моделювання послідовного робочого потоку великої діяльності, орієнтуючись на послідовності дій та відповідні умови ініціювання дії. Стан діяльності стосується виконання кожного кроку та процесу.

Діаграма діяльності представлена фігурами, з'єднаними стрілками. Стрілки проходять від початку діяльності до завершення і представляють послідовний порядок виконуваних дій. Чорні кола представляють початковий стан робочого процесу. Обведений чорним колом позначає кінцевий стан. Закруглені прямокутники представляють виконані дії, які описуються текстом всередині кожного прямокутника.

У випадку даної системи діаграма діяльності буде представлена для відображення процесу дій користувача з вибору товару та оформлення завомлення. За допомогою цієї системи і буде розглянуто та створено автоматизований тест. Діаграма діяльності зображена на рис 2.2.

Рисунок 2.2 – Діаграма діяльності

2.2.3 Діаграма пакетів

Важливим завданням систематизації інформації про предметну область є розбиття великої системи на невеликі підсистеми. Саме тут особливо помітні структурні та об'єктно-орієнтовані відмінності між підходами. Одна з ідей полягає у групуванні класів у компоненти вищого рівня. У UML такий механізм угруповання зветься пакетів (package). Діаграмою пакетів є діаграма, що містить пакети класів та залежності між ними. Строго кажучи, пакети є елементами діаграми класів, тобто діаграма пакетів – це лише діаграма класів. Відрізняються ці діаграми практичним призначенням та

використанням. Залежність між двома елементами має місце в тому випадку, якщо зміни у визначенні одного елемента можуть спричинити зміни в іншому.

Пакети є життєво важливими для великих проектів. Особливо коли діаграма класів, що охоплює всю систему важко читати. Пакети не дають відповіді на питання, яким чином можна зменшити кількість залежностей у системі, що розробляються, вони допомагають виділити ці залежності. Пакети корисні під час тестування системи. Кожен пакет може містити один або кілька класів, що тестуються, за допомогою яких перевіряється поведінка пакета.

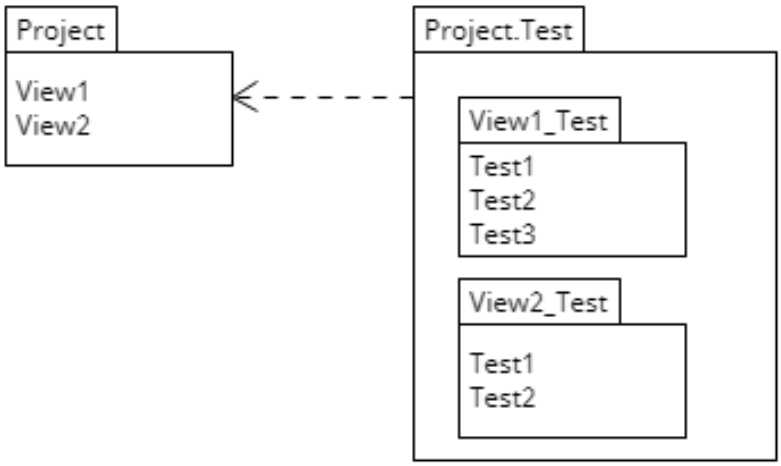


Рисунок 2.3 – Діаграма пакетів

2.2.4 Діаграма компонентів

Побудова діаграми компонентів – наступний етап об'єктно-орієнтованого аналізу та проектування. Для побудови діаграми компонентів необхідно мати побудовані діаграми прецедентів, діаграми класів та діаграми послідовності (і (або) діаграми кооперації).

Діаграма компонентів (Component Diagram) показує структурні компоненти ІС та зв'язку між ними. Як компоненти розглядаються лише інформаційні об'єкти: файли, модулі, бібліотеки, пакети тощо.

Компонент - це фізично існуюча частина системи, завдяки якій забезпечується реалізація класів і відносин, а також досягається функціональність ІС, що моделюється.

Для компонентів UML визначає такі стереотипи:

1) file {файл) — найпоширеніший різновид компонента, що набуває вигляду будь-якого файлу;
2) executable (здійсненний) - різновид файлу, що є здійсненним; може виконуватись на будь-якій комп'ютерній платформі;
3) document (документ) — різновид файлу у форматі документа, який не є здійсненним і не містить вихідний код програми;
4) library (бібліотека) – різновид файлу у форматі бібліотеки (динамічної або статичної);
5) source (джерело) — різновид файлу, який містить вихідний код програми;
6) table (таблиця) - різновид компонента у форматі таблиці БД.

Компонент зображується у вигляді великого прямокутника, ліворуч на якому розташовані два маленькі витягнуті вздовж прямокутника. Графічне зображення компонента походить від зображення модуля системи.

Інтерфейс – наступний елемент діаграми компонентів. Інтерфейс відображається у вигляді кола (не обов'язково замкненого), яке пов'язане з компонентом певним ставленням.

На Рисунку 2.4 зображено діаграму компонентів даного проекту.

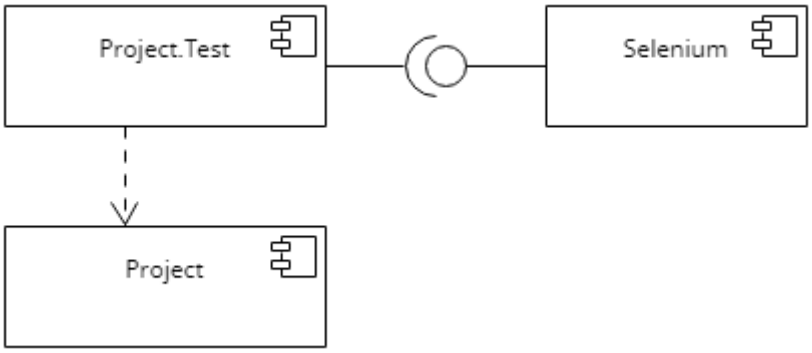


Рисунок 2.4 – Діаграма компонентів

2.3 Структура даних в системі

Так як автоматизований тест буде орієнтуватися на структуру даних веб-сайту, то і буде показано структуру веб-сайту.

PageObject – це шаблон проектування, який поділяє високорівневу логіку від низькорівневої логіки пошуку та використання елементів інтерфейсу, при цьому підвищується читання та підтримка коду. В результаті виходить окремий клас, який описує, конкретну сторінку з її веб-елементами, і навіть методи взаємодії із нею.

Ця структура представлена рисунку нижче:

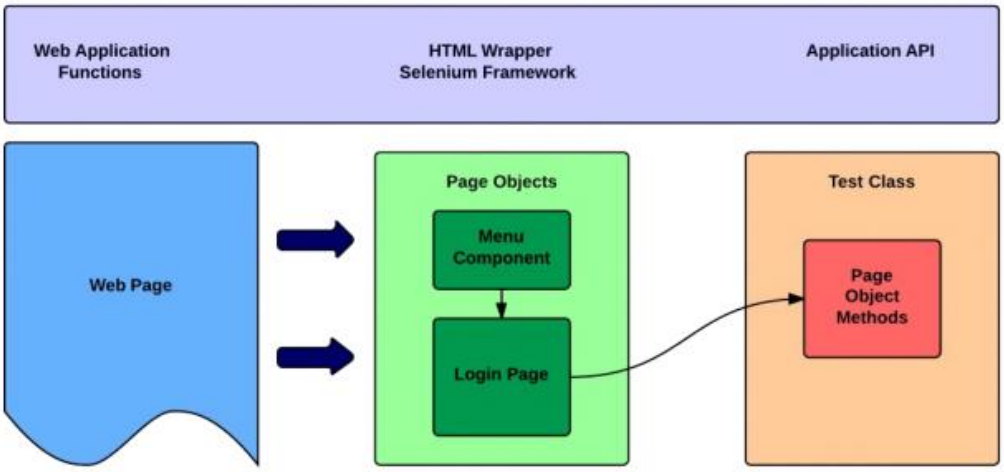


Рисунок 2.5 - Структура патерна PageObject

2.3.1 Структура XML

Синтаксично у XML, у порівнянні з HTML, немає нічого нового. Це такий самий текст, розмічений тегами, але з тією різницею, що в HTML існує обмежений набір тегів, які можна використовувати в документах, в той час як XML дозволяє створювати та використовувати будь-яку розмітку, яка може знадобитися для розмітки даних.

Безсумнівною перевагою XML є те, що це досить проста мова. Основних конструкцій у XML мало, але, незважаючи на це, за допомогою їх можна створювати розмітку документів практично будь-якої складності.

Для демонстрації структури XML документа краще звернутися до якогось прикладу. Розглянемо наступний XML документ:

```
<?xml version="1.0" ?>
<кореневий_елемент>
     <елемент>
         Текст <ще_элемент атрибут="значення" /> текст ...
     </елемент>
     Текст текст текст <елемент>текст текст... </елемент>  ...
</кореневий_элемент>
```

Рисунок 2.6 – Структура XML

Перший рядок документа визначає його як XML документ, побудований відповідно до першої версії мови (version="1.0"). У цій же конструкції можна вказати і кодування, в якому створено документ:

xml version="1.0" encoding="Windows-1251" ?>.

Кодування за замовчуванням для XML є unicode. Далі знаходиться тег кореневого (головного) елемента <кореневий_елемент>, що містить елемент <елемент>, який, у свою чергу, містить елемент <ще_елемент атрибут="значення" /> з атрибутом атрибут. Як видно з прикладу, правила запису елементів, атрибутів та їх значень в XML нічим не відрізняються від правил запису елементів атрибутів та їх значень у HTML (також є теги елементів, що відкривають і закривають, елементи з вмістом і без і т.д.), тільки набір елементів дещо розширений, завдяки чому ми можемо "навантажити" розмітку семантикою.

Нижче наведено кілька правил побудови документа XML. Отже:

1) будь-який XML документ повинен починатися рядком <?xml version="1.0" ?>
2) будь-який XML документ повинен мати єдиний (не більше, не менше!) кореневий елемент; наприклад, HTML для цих цілей використовувався елемент <html>, у прикладі вище - це <кореневий_елемент>.
3) кодуванням за замовчуванням для символів XML документа є Unicode кодування UTF-8, тому XML файли повинні бути збережені у відповідному кодуванні або в 1-му рядку документа має бути задане кодування документа, наприклад encoding="Windows-1251" (при роботі тільки з латиницею це ніяк не виявляє, оскільки кодування цих символів в ASCII збігається з UTF-8).
4) правила запису більшості конструкцій мови співпадає з правилами XHTML, що вивчався вами раніше (детальніше мова про основні конструкції мови піде далі в уроці).

XML документ є звичайним текстовим файлом з розширенням .xml. Єдина особливість їх у тому, що з символів файла рекомендується використовувати кодування Unicode.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ДОДАТКУ

3.1 Набір інструментів використаних для розробки

Розберемо інструменти, що використовуються на практиці для написання тестових скриптів.

Для будь-якої розробки необхідне спеціальне середовище. Для цього використовується інтегровані середовища розробки (Integrated development environment – IDE) – комплекс програмних засобів, що використовується програмістами для розробки програмного забезпечення (ПЗ) або для автоматизації тестування.

Середовище розробки включає:

1) Текстовий редактор;
2) Компілятор чи інтерпретатор;
3) Кошти автоматизації складання;
4) Відладчик.

У світі існує велика кількість IDE від різних відомих компаній. Вивчаючи особливості мови програмування C#, для написання автотестів було вибрано середовище розробки Visual Studio.

Microsoft Visual Studio - це програмне середовище з розробки додатків для ОС Windows, як консольних, так і з графічним інтерфейсом.

У комплект входять такі основні компоненти:

1) Visual Basic.NET – для розробки додатків на VisualBasic;
2) Visual C++ - традиційною мовою C++;
3) Visual C# - мові C# (Microsoft);
4) Visual F# - F# (Microsoft Developer Division).

Функціональна структура середовища включає:

1) редактор вихідного коду, який включає безліч додаткових функцій, як автодоповнення IntelliSense, рефракторинг коду тощо;
2) налагоджувач коду;
3) редактор форм, призначений для спрощеного конструювання графічних інтерфейсів;
4) веб-редактор;
5) дизайнер класів;
6) дизайнер схем баз даних.

Visual Studio також дозволяє створювати та підключати сторонні доповнення (плагіни) для розширення функціональності практично на кожному рівні, включаючи додавання підтримки систем контролю версій вихідного коду (Subversion та VisualSourceSafe), додавання нових наборів інструментів (для редагування та візуального проектування коду на предметно-орієнтованих мовах) програмування або інструментів інших аспектів процесу розробки програмного забезпечення).

Комерційні версії у порядку зростання ціни: Visual Studio Professional, Visual Studio Premium та Visual Studio Ultimate.

Переваги та недоліки

Інтегроване середовище розробки (IntegratedDevelopmentEnvironment - IDE) Visual Studio пропонує ряд високорівневих функціональних можливостей, що виходять за рамки базового керування кодом.

Visual Studio також має безліч інших функцій: можливість управління проектом; вбудована функція керування вихідним кодом; можливість рефакторизації коду; потужна модель розширюваності. Більше того, у разі

використання Visual Studio 2008 Team System розробник отримує розширені можливості для модульного тестування, спільної роботи та управління версіями коду (що значно більше того, що пропонується у більш простих інструментах типу Visual SourceSafe).

Як недолік можна відзначити неможливість відладчика (Microsoft Visual Studio Debugger) відстежувати у коді режиму ядра. Налагодження в Windows у режимі ядра в загальному випадку виконується під час використання WinDbg, KD або SoftICE.

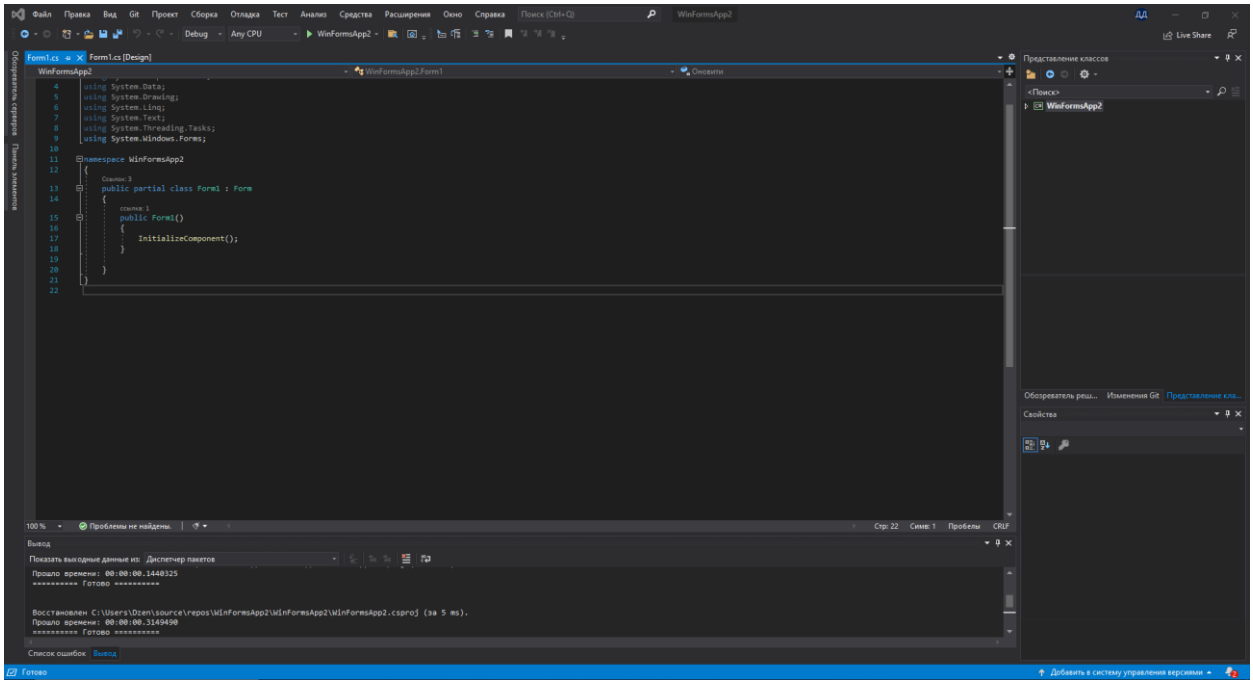


Рисунок 3.1 – Інтерфейс Visual Studio

Для тестування веб-додатків та емулювання роботи користувача в браузері був обраний популярний інструмент Selenium, а саме Selenium WebDriver. За призначенням Selenium WebDriver є драйвер браузера, тобто програмну бібліотеку, яка дозволяє розробляти програми, що керують поведінкою браузер. Найбільш популярною сферою застосування Selenium є автоматизація тестування веб-додатків. Однак за допомогою Selenium можна

і навіть потрібно автоматизувати будь-які інші рутинні дії, виконувані через браузер.

Розробка Selenium підтримується виробниками популярних браузерів: Chrome, Opera, Safari, Mozilla, Internet Explorer. Виробники браузерів адаптують їх для більш тісної інтеграції з Selenium, а іноді навіть реалізують вбудовану підтримку Selenium у самому браузері. Selenium є центральним компонентом цілого ряду інших інструментів та фреймворків автоматизації.

Selenium підтримує десктопні та мобільні браузери. Selenium дозволяє розробляти сценарії автоматизації практично на будь-якому мовою програмування. За допомогою Selenium можна організовувати розподілені стенди, що складаються із сотень машин з різними операційними системами та браузерами, і навіть виконувати сценарії в хмар. Приклад використання Selenium WebDriver представлений на рисунку

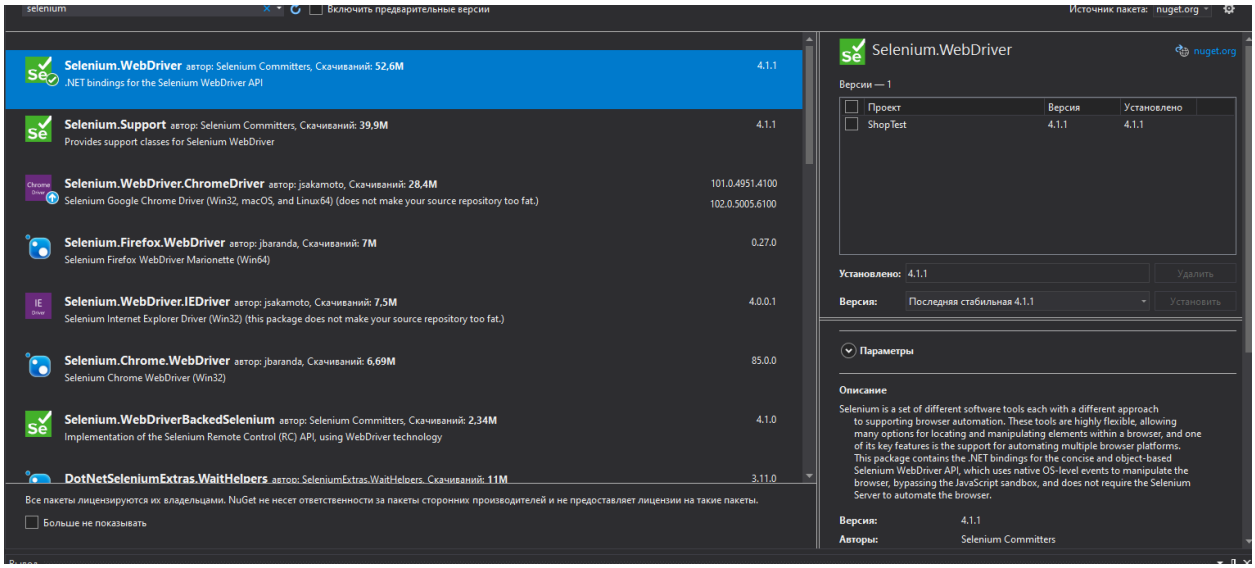


Рисунок 3.2 – Вікно завантаження бібліотек

3.2 Функціонал коду

Функціонал коду представлений у вигляді автоматизованого тестування інтернет-магазину, тому в ньому задіяні конкретні селектори пов’язані з веб-ресурсом та їх пошук. Нижче буде перераховано та описано основні типи селекторів.

Багато браузерів реалізують основу CSS, щоб розробники змогли використовувати CSS таблиці у своїх проектах. Що дозволяє розділити між собою контент сторінки з оформленням. У CSS є патерни, за якими стилі, створювані розробником, застосовуються до елементів сторінки (DOM). Ці патерни називаються локаторами (selectors). Selenium WebDriver використовує той самий принцип для знаходження елементів. І він набагато швидше, ніж пошук елементів на основі XPath,

XPath (XML path) – це мова для нодів (nodes) у XML документі. Оскільки багато браузери підтримують XHTML, ми можемо використовувати XPath для знаходження елементів на web сторінках.

Важливою відмінністю між CSS і XPath локаторами є те, що, використовуючи XPath, ми можемо переміщуватися як у глибину DOM ієрархії, так і повертатися назад. Що ж до CSS, то тут ми можемо рухатися лише у глибину. Це означає, що з XPath можемо знайти батьківський елемент, за дочірнім.

Також існують пошуки за id, href, class і т.д.. В даному проекті використано пошукт за допомогою селектора XPath, так як він краще здатний до роботи з динамічними типами сайтів. Приклад використання селектора показано на рис 3.3.

```
private readonly By _FindProduct = By.XPath("//button[text()=' Найти ']");
private readonly By _ProductPage = By.XPath("//a[@title='Компьютер Artline Gaming X51 v07 (X51v07) ']");
private readonly By _ProductBuy = By.XPath("//button[@aria-label='Купить']");
```

Рисунок 3.3 – Селектор XPath

На рисунку вище показано що пошук по XPath йде по різним тегам сайту: Button, text, title, class.

Автоматизоване тестування сайту створено по прикладу діаграми діяльності, тобто один із варіантів використання системи користувачем. На рисунку 3.4 ми бачимо відкриття сайту та збільшення його розміру на весь екран, а також підключення драйверу selenium для роботи в браузері.

```
driver = new OpenQA.Selenium.Chrome.ChromeDriver();
driver.Navigate().GoToUrl("https://rozetka.com.ua/");
driver.Manage().Window.Maximize();
```

Рисунок 3.4 – Підключення драйверу, та вхід на сайт

На рисунку 3.5 та 3.6 відповідно показано очікування системи з подальшим введенням даних в поле пошуку та натиснення клавіші “пошук”.

```
Thread.Sleep(500);
var SearchProduct = driver.FindElement(_SearchProduct);
SearchProduct.SendKeys("artline");
```

Рисунок 3.5 – Запис тексту в поле

```
Thread.Sleep(500);
var FindProduct = driver.FindElement(_FindProduct);
FindProduct.Click();
```

Рисунок 3.6 – Натиснення на клавішу

Весь код побудований на такого типу архітектурі, тобто послідовності дій від входу на сайт, до підтвердження замовлення в кошику з введенням всіх даних користувача. Фрагмент коду з очікуванням потрібний для

правильного функціонування тесту на динамічному типу сайтів яким і являється інтернет-магазин.

4 ВИКОРИСТАННЯ ТА ТЕСТУВАННЯ СИСТЕМИ

4.1 Опис роботи автоматизованого тесту

Для початку потрібно встановити розширення в браузер Google Chrome, щоб написаний тест зміг підключитися до системи браузера та виконувати в ньому дії. Додати його можна через інтернет-магазин Chrome у вкладці «Розширення» (рис 4.1).

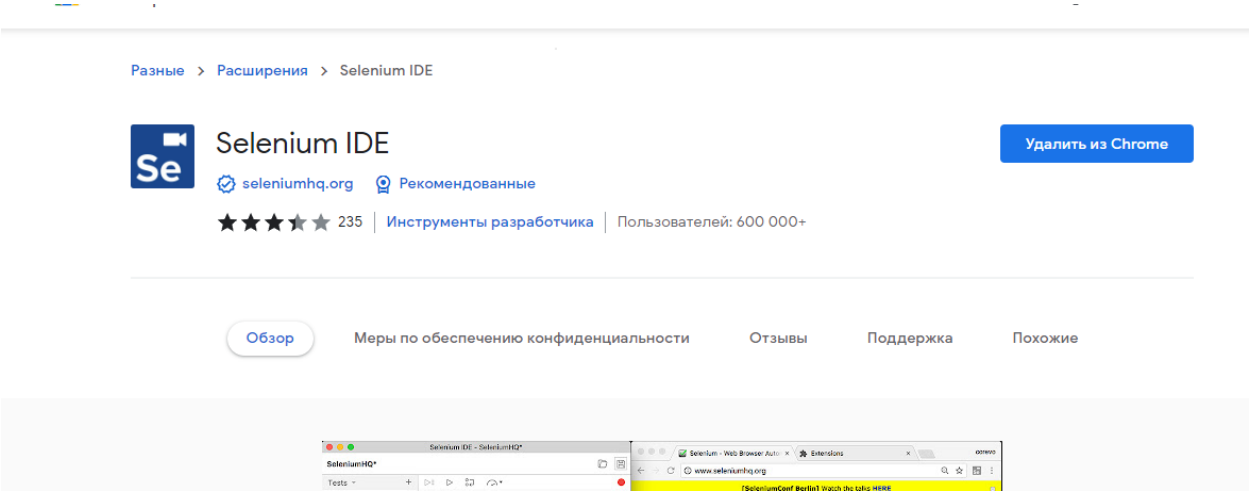


Рисунок 4.1 – Вікно додання розширення

Додаток вже встановлений і для роботи автоматизованого тесту потрібна лише наявність цього розширення.

Для запуску тесту потрібно виконати певні дії в Visual Studio, а саме відкрити вкладку «Тест» та вибрати «Оглядач тестів».

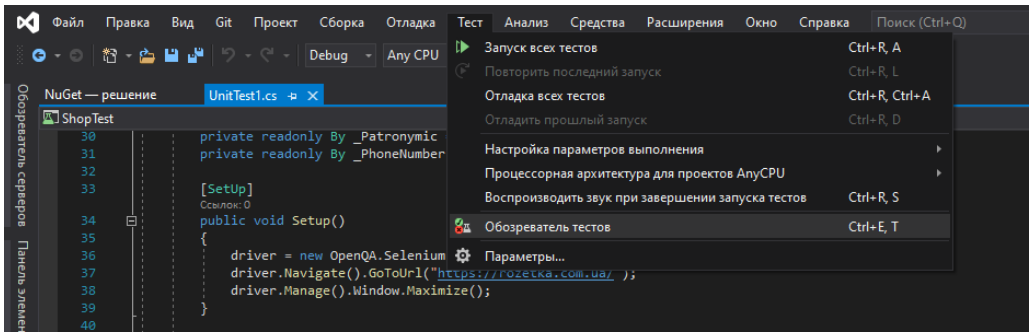


Рисунок 4.2 – Вибір оглядача тексту

Далі відкриється «Оглядач тестів» в якому буде вказано статус виконання тесту та інші деякі його параметри.

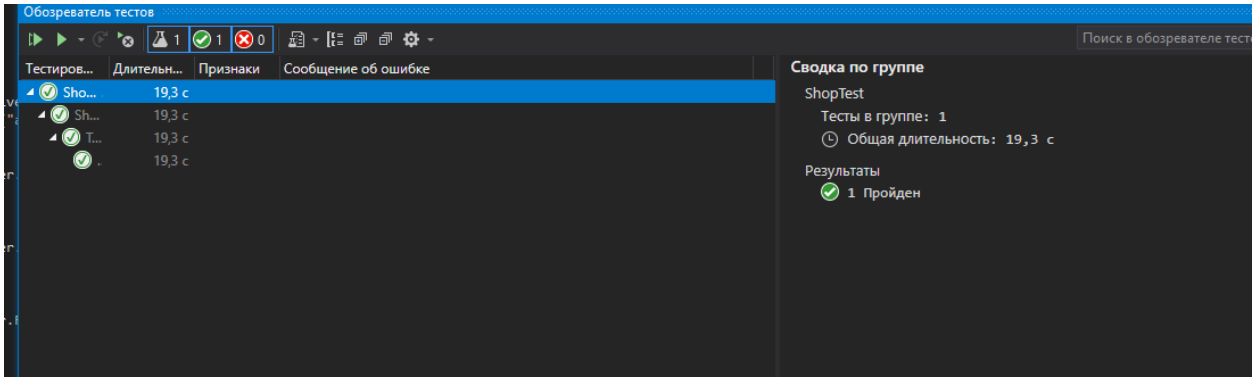


Рисунок 4.3 – Оглядач тестів

Потрібно натиснути клавішу запуску тесту, і він почне виконуватися. Далі відкриється браузер та почнуть виконуватися дії вказані в коді, також в браузері буде додатково показано статус автоматизованого тесту:

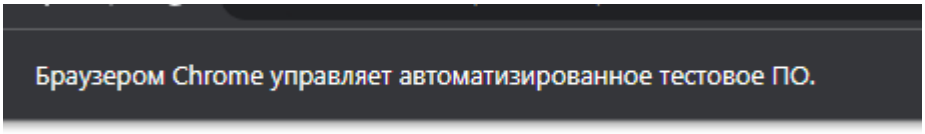


Рисунок 4.4 -Текст про виконання автоматизованого тесту

Після закінчення тесту в «Оглядачі тестів» буде показано наступний статус:

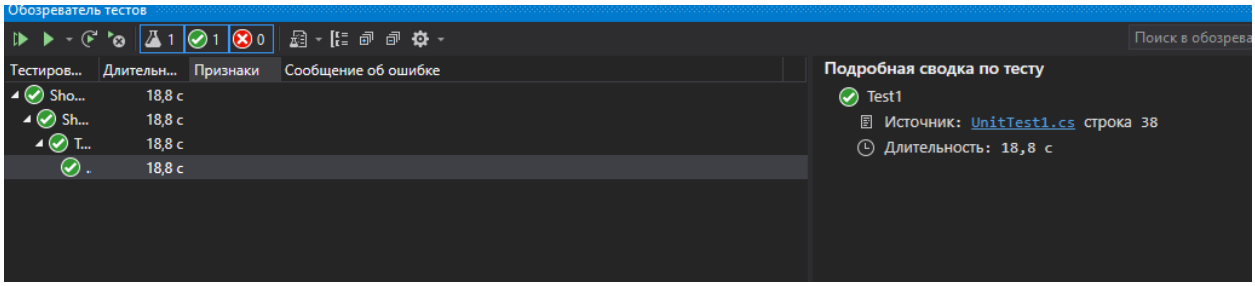


Рисунок 4.5 -Статус автоматизованого тесту

Тут можна побачити стан та час виконання тесту.

4.2 Результат тестування

Процес виконання автоматизованого тесту на статичній та динамічній сторінці досить відрізняється. В той час як на статичній нічого не змінюється і тест буде мати досить правильний вигляд, на динамічному можуть виникнути проблеми зі зміною структури сторінки або переходу на інші ресурси. Він також може включати в себе роботу з JS скриптами та іншими додатковими елементами архітектури сайтів. Автоматизований тест в даній дипломній роботі побудований на основі динамічного сайту.

При першому проходженні тесту було виявлено проблему, на якій було вказано що тест не може знайти елемент:

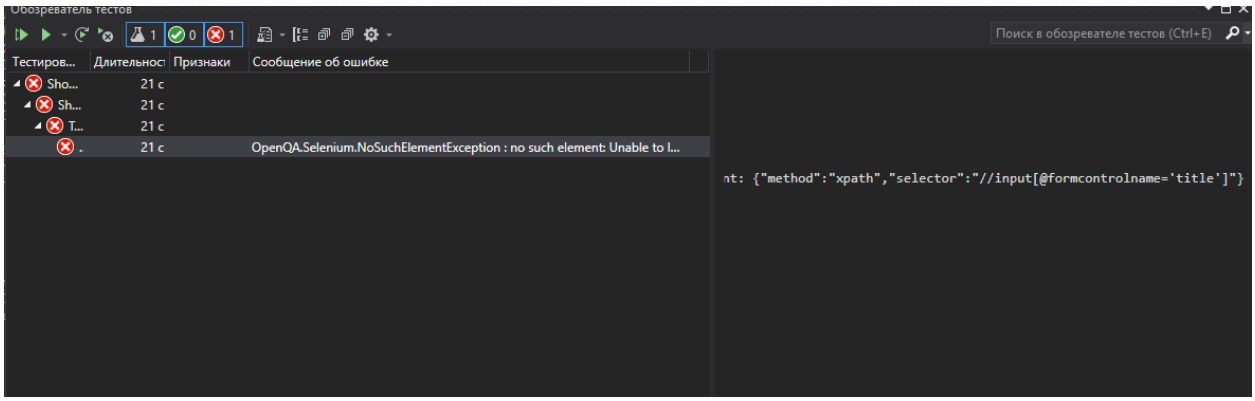


Рисунок 4.6 – Помилка при виконанні

Ця проблема відбувається із-за того що автоматизований тест виконує дію яка ще не з’явилася на сторінці, і перестає функціонувати. Ця проблема вирішилась доданням частини коду Thread.Sleep (кількість). Воно відповідає за очікування системи – скільки система має очікувати до виконання подальших дій. Додання цього фрагменту з різною кількістю очікування потребувалось в деяких кроках виконання автоматизованого тесту

Висновки

В даній роботі було проведено аналіз сутності веб-додатків, технік та методологій їх тестування у життєвому циклі розробки програмного забезпечення, внаслідок чого було досліджено основи роботи інтернет магазину, структури веб-сторінок, варіанти пошуку веб-елементів та способів їх тестування.

1) Було виявлено, що функціональне (ручне) тестування вигідніше застосовувати при розробці нового бізнес-процесу, а автоматизоване рішення доцільніше використовувати для перевірки вже існуючого функціоналу.

2) На основі аналізу документації та технік тестування були написані набори тестових випадків, які дозволили перевірити працездатність процесу формування кошику покупця та оформлення замовлення.

3) Проведено дослідження сутності автоматизованого тестування веб-застосунків. Після чого були розроблені автоматизовані сценарії тестування вже існуючого функціоналу інтернет-магазину. Це дозволило отримати позитивний економічний ефект, значно скоротивши час та витрати на ручне тестування, а також виключити вплив людського фактора у процесі тестування.

4) Розглянуто основні проблеми при роботі зі статичними та динамічними сайтами, та шляхи їх вирішення.

5) Описано програмні засоби та інструменти, котрі було застосовано для розробки програмного забезпечення. Виявлено що середовище розробки Visual Studio гарно для цього підходить.

ПЕРЕЛІК ПОСИЛАНЬ

1. Web-Lighthouse [Електронний ресурс] – Режим доступу: https://web-lighthouse.com
2. IntelliJ IDEA [Електронный ресурс]. – Режим доступу: https://jetbrains.ru/products/idea/
3. Selenium webdriver [Електронный ресурс]. – Режим доступу: https://www.seleniumhq.org/projects/webdriver/
4. Apache Maven [Електронный ресурс]. – Режим доступу: https://maven.apache.org
5. Ручне та автоматизоване тестування. [Електронний ресурс] – Режим доступу: https://qalight.ua
6. Криспин, Л. Гибкое тестирование. Практическое руководство для тестировщиков ПО и гибких команд / Лиза Криспин, Джанет Грегори: пер. с англ. – Н. Мухин; Вильямс, 2016. – 464 с.
7. Блэк, Р. Ключевые процессы тестирования / Рэкс Блэк – 3-е изд.; пер. с англ. – Р. Павлов – М.: Издательство «Лори», 2004. – 537 с.
8. Тестирование Дот Ком, или Пособие по жестокому обращению с багами в интернет-стартапах, Савин Р., 2007.

ДОДАТОК А

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра Інженерії програмного забезпечення

«РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ З АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ІНТЕРНЕТ-МАГАЗИНУ МОВОЮ C#»

Виконав студент
Групи ПД-44
Дзененко Владислав Олегович
Керівник роботи: ст.в. Гаманюк Ігор Михайлович

Київ 2022

1

Технічне завдання

1) Створення тестових сценарії для автоматизації найбільш використовуваних сценаріїв роботи користувача на даному ресурсі.
2) Перевірити правильне функціонування інтернет-магазину та відсутність багів, або їх пошук.
3) Забезпечити ефективність виконання автоматизованого тестів.

2

Об’єкт, предмет та мета роботи

- **Мета дослідження** – програмне забезпечення тестування застосунку
- **Об’єкт дослідження** – тестування застосунку
- **Предмет дослідження** – підвищення швидкості процесу тестування

3

Порівняння аналогів інструментів тестування

Selenium	Postman
Особливості інструменту:	Особливості інструменту:
Гнучкість	Повний набір функцій для розробки, налагодження, тестування, документування та публікації API
Працює на всіх платформах та основних браузерах	Зручний і простий у використанні інтерфейс користувача
Можна писати на любій мові	Підтримка як автоматизованого, так і дослідницького тестування
Потрібні навички в написанні скриптів та програмуванні	

4

Порівняння аналогів інструментів тестування

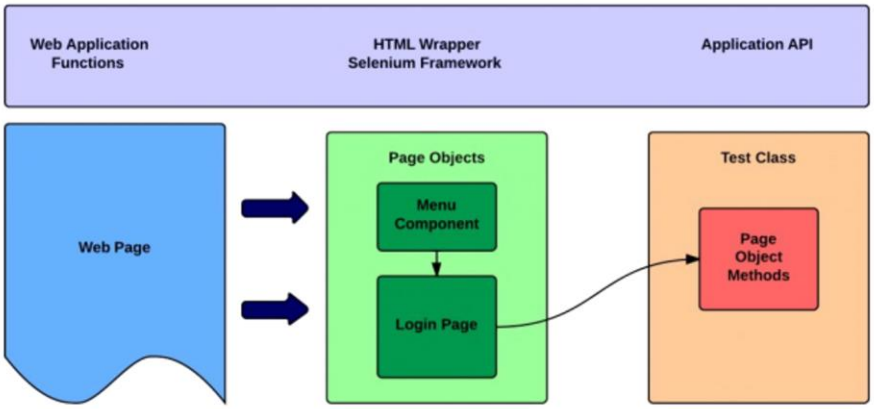
SoapUI	Cypress
Особливості інструменту:	Особливості інструменту:
Генерування тестів простим перетягуванням, за допомогою вказівки та кліку	Юніт-тестування
Ефективне тестування з використанням даних із файлів та баз даних.	Інтеграційне тестування
Асинхронне тестування	Комплексне тестування
Скрипти можуть використовуватись повторно	Працює на Javascript

5

Програмні та технічні засоби

6

Архітектура веб-сторінок



7

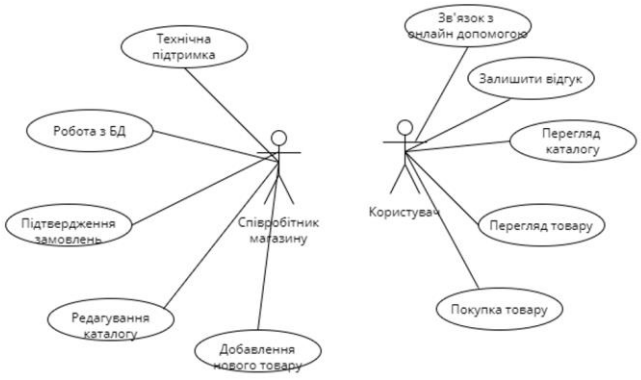
Функціональність інтернет-магазину

8

Архітектура автоматизованого тесту

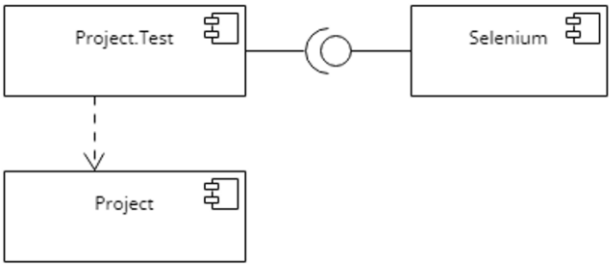
10

Діаграма прецедентів



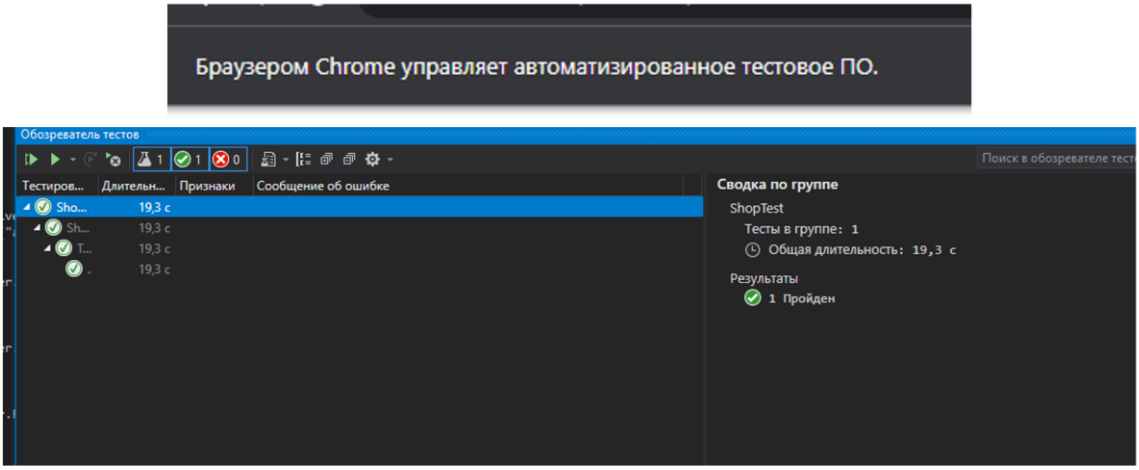
9

Діаграма компонентів



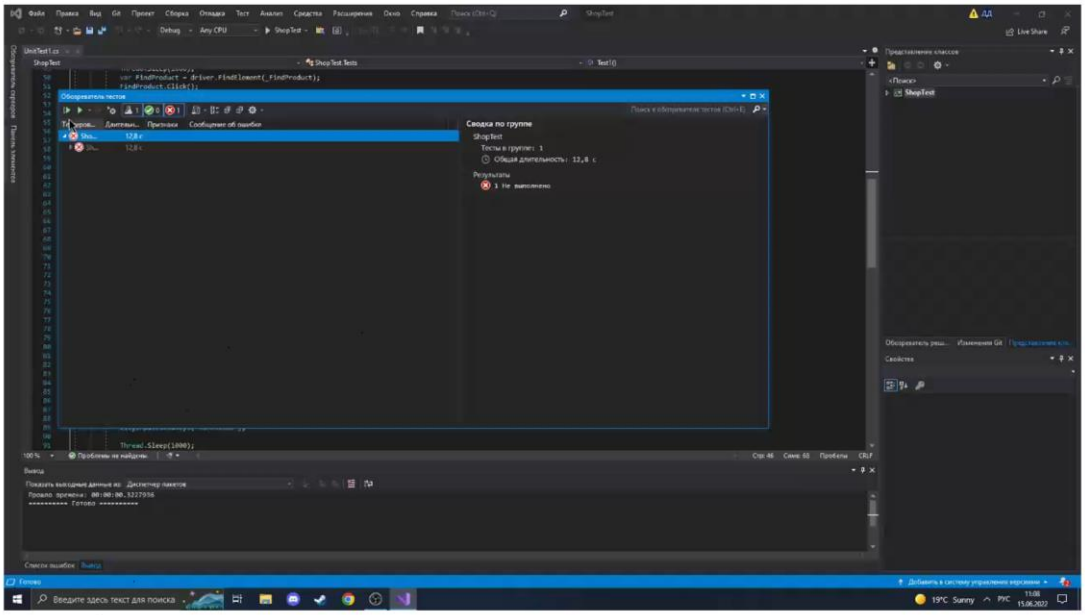
11

РОЗРОБКА АВТОМАТИЗОВАНИХ СЦЕНАРІЇВ ТЕСТУВАННЯ ІНТЕРНЕТ-МАГАЗИНУ



12

РОБОТА АВТОМАТИЗОВАНИХ СЦЕНАРІЇВ ТЕСТУВАННЯ ІНТЕРНЕТ-МАГАЗИНУ



13

Висновки

1. Виявлено, що функціональне (ручне) тестування вигідніше застосовувати при розробці нового бізнес-процесу, а автоматизоване рішення доцільніше використовувати для перевірки вже існуючого функціоналу.

2. На основі аналізу документації та технік тестування були написані набори тестових випадків, які дозволили перевірити працездатність процесу формування кошику покупця та оформлення замовлення.

3. Проведено дослідження сутності автоматизованого тестування веб-застосунків. Після чого були розроблені автоматизовані сценарії тестування вже існуючого функціоналу інтернет-магазину. Це дозволило отримати позитивний економічний ефект, значно скоротивши час та витрати на ручне тестування, а також виключити вплив людського фактора у процесі тестування.

4. Розглянуто основні проблеми при роботі зі статичними та динамічними сайтами, та шляхи їх вирішення.

5. Описано програмні засоби та інструменти, котрі було застосовано для розробки програмного забезпечення. Виявлено що середовище розробки Visual Studio гарно для цього підходить.

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1) Дзененко В.О., Гаманюк І.М. Функціональне та автоматизоване тестування інтернет-магазину / НАУКОВО-ТЕХНІЧНА КОНФЕРЕНЦІЯ "ЗАСТОСУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ В ІНФОКОМУНІКАЦІЙНИХ ТЕХНОЛОГІЯХ" Збірник тез 20.04.2022, ДУТ, м. Київ – К.: ДУТ, 2022.

2) Дзененко В.О., Гаманюк І.М. Використання UML діаграми прецедентів для моделювання основних функцій інтернет-магазину / НАУКОВО-ТЕХНІЧНА КОНФЕРЕНЦІЯ "ЗАСТОСУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ В ІНФОКОМУНІКАЦІЙНИХ ТЕХНОЛОГІЯХ" Збірник тез 20.04.2022, ДУТ, м. Київ – К.: ДУТ, 2022.

15

Дякую за увагу!

16