

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
„КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського”

Факультет прикладної математики
Кафедра програмного забезпечення комп’ютерних систем

КУРСОВА РОБОТА

з дисципліни «Компоненти програмної інженерії».

Частина 4. Якість та тестування програмного забезпечення»

на тему

**Розробка програмного продукту
«Димове тестування авторської програми»**

Виконала студентка

II курсу групи КП-11

Кирильчук Олександра

Артурівна

Керівник роботи

доцент, к.т.н. Заболотня

Т.М. Оцінка

(дата, підпис)

КИЇВ 2023

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ.....	3
ВСТУП	4
1. АСПЕКТИ СУЧАСНИХ ПІДХОДІВ ТА МЕТОДІВ ДИМОВОГО ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	6
1.1 Сутність поняття тестування програмного забезпечення	6
1.2 Автоматизоване тестування	7
1.3 Димове тестування.....	10
1.4 Web-додатки	13
1.5 Характеристика інструменту Selenium	16
2. РОЗРОБКА ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	19
2.1 Опис Web-додатку.....	19
2.2 Необхідні інструменти для створення автотестів	20
2.3 Вибір мови програмування.....	21
2.4 Створення документації.....	23
3. КІНЦЕВІ РЕЗУЛЬТАТИ.....	24
3.1 Автоматизоване димне тестування системи веб-додатку.....	24
3.1.1. Створення проєкту	24
3.1.2. Створення тест-кейсів.....	26
3.2 Аналіз отриманих результатів	35
ВИСНОВКИ.....	37
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	39

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

ПК – персональний комп'ютер.

ПЗ – програмне забезпечення.

IDE – середовище розробки.

ОС – операційна система.

HTTP – Hypertext Transfer Protocol.

HTTPS – HyperText Transfer Protocol Secure.

API – прикладний програмний інтерфейс.

GUI – графічний інтерфейс користувача.

XML – розширювана мова розмітки (Extensible Markup Language).

Selenium – набір інструментів, призначених для автоматизації веб браузерів на різних платформах.

ВСТУП

Тестування програмного забезпечення є важливою частиною процесу розробки. Воно допомагає гарантувати, що програмне забезпечення відповідає вимогам і не містить помилок.

У 2023 році тестування програмного забезпечення (ПЗ) продовжує розвиватися в напрямку автоматизації, штучного інтелекту та хмарних технологій.

Автоматизація тестування є однією з основних тенденцій тестування в останні роки. Вона дозволяє виконувати тести швидко і ефективно, що є особливо важливим для великих і складних проєктів. У 2023 році автоматизація тестування буде продовжувати розвиватися завдяки появі нових інструментів і технологій. Наприклад, машинне навчання і штучний інтелект будуть використовуватися для створення більш ефективних і точних тестів.

Одним із видів тестування програмного забезпечення є димове тестування. Димове тестування - це швидке і поверхневе тестування, яке проводиться для перевірки основних функцій програмного забезпечення. Димове тестування зазвичай виконується на ранніх етапах розробки, щоб визначити, чи готова програма до подальшого тестування.

У цій курсовій роботі буде проведено димове тестування авторської програми, а саме веб-застосунку “ToDo” за допомогою інструменту Selenium WebDriver та Selenium IDE

В роботі буде використано мову програмування Java, оскільки вона має зручний синтаксис, широкі можливості для обробки об'єктів та велику кількість наявних бібліотек.

Об'єктом дослідження є димове тестування веб-додатків.

Мета даної курсової роботи – проаналізувати теорію автоматизованого smoke тестування веб-додатків та створити автотести на основі тест-кейсів за допомогою інструменту Selenium WebDriver та Selenium IDE.

Для досягнення цієї мети необхідно вирішити такі завдання:

- Провести аналіз теоретичних основ димового тестування.
- Розробити теск-кейси димового тестування для веб-застосунку “ToDo”.
- Виконати димове тестування веб-застосунку “ToDo” за допомогою інструменту Selenium WebDriver та Selenium Remote Control.
- Проаналізувати результати димового тестування.

1. АСПЕКТИ СУЧАСНИХ ПІДХОДІВ ТА МЕТОДІВ ДИМОВОГО ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Сутність поняття тестування програмного забезпечення

Тестування – це одна з технік контролю якості, що включає в себе планування, проєктування, виконання тестування та аналіз відповідних результатів.

Тестування програмного забезпечення – це процес аналізу програмного засобу і відповідної документації, з метою виявлення дефектів і підвищення якості продукту.[4]

Тестування ПЗ охоплює не тільки проведення тестів, але й інші елементи процесу забезпечення якості, такі як:

1. Планування та аналіз вимог.
2. Критерії початку тестування.
3. Стратегія тестування.
4. Проєктування тестових сценаріїв (тест-планів, тест-кейсів, користувацьких вимог).
5. Виконання тест-кейсів, вимог.
6. Фіксація дефектів.
7. Аналіз результатів.
8. Написання звітів.

Стосовно цілі тестування, можна виділити такі основні аспекти:

- Надання інформації про якість програмного забезпечення кінцевому замовнику.
- Підвищення якості тестового ПЗ.
- Запобігання виявленню нових дефектів.

Для професійного тестування використовується тест-плани з варіантами тестування.

Test Plan – це документ, який описує весь розмір роботи, яку повинен виконати тестувальник, починаючи з опису об'єкту, цілі, ресурси і графіки запланованих тестових активностей, стратегії тестування, розкладу, критерії початку і кінця тестування, до необхідного в процесі тестування обладнання, методи проєктування тестів. Оцінка всіх можливих ризиків тестування з варіантами їх можливого вирішення. [4]

Test Case - це описана послідовність певних дії (кроків) і очікуваний результат для перевірки роботи певного функціоналу системи. Також необхідно, щоб опис кейса був таким, щоб виконати його міг будь-хто (тестувальник, розробник, аналітик, замовник).

1.2 Автоматизоване тестування

Автоматизація тестів – найкращий спосіб підвищити ефективність, охоплення тестом і швидкість виконання при тестуванні програмного забезпечення. [1]

Автоматизоване тестування програмного забезпечення важливо з таких причин:

- Ручне тестування всіх робочих процесів, усіх полів, усіх негативних сценаріїв вимагає часу та грошей.
- Складно перевірити багатомовні сайти вручну.

- Автоматизація тестів при тестуванні програмного забезпечення не вимагає втручання людини. Ви можете запустити тест і залишити його без нагляду.
- Автоматизація тестів збільшує швидкість виконання тесту.
- Автоматизація сприяє збільшенню охоплення тестом.
- Тестування вручну може стати нудним, схильним до помилок.

У процесі автоматизації виконуються наступні кроки (рис. 1.2):

- 1) Вибір інструменту тестування.
- 2) Визначення сфери автоматизації.
- 3) Планування, проєктування та розробка.
- 4) Виконання тесту.
- 5) Технічне обслуговування.

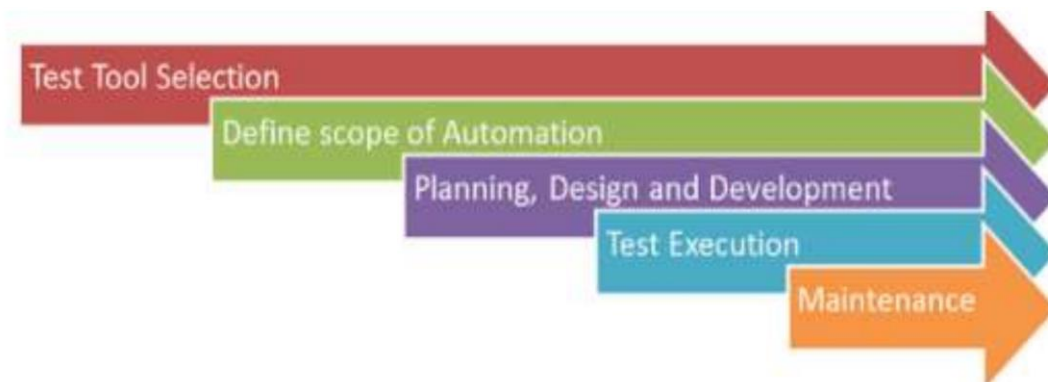


Рис. 1.2. Процеси автоматизованого тестування.

Види автоматизованого тестування:

- Димове тестування.

Тестування, що проводиться на початковому етапі і в першу чергу спрямоване на перевірку готовності розробленого продукту до проведення більш розширеного тестування, визначення загального стану якості продукту.

- Модульне тестування.

Це тестування кожної атомарної функціональності додатку окремо, в штучно створеному середовищі. Модульне тестування проводиться під час розробки програми розробниками.

- Інтеграційне тестування.

Вид тестування, при якому на відповідність вимог перевіряється інтеграція модулів, їх взаємодія між собою, а також інтеграція підсистем в одну загальну систему. Метою цього рівня тестування є виявлення дефектів взаємодії між цими програмними модулями при їх інтеграції.

- Функціональне тестування.

Один із видів тестування, спрямованого на перевірку відповідностей функціональних вимог ПЗ його реальним характеристикам. Основним завданням функціонального тестування є підтвердження того, що програмний продукт, який розробляється, володіє усім необхідним замовнику функціоналом.

- Регресійне тестування.

Визначається як тип тестування програмного забезпечення для підтвердження того, що нещодавня зміна програми або коду не вплинула негативно на існуючі функції.

- Перевірка рівня даних.

Сконцентровано на тій частині програми, яка відповідає за зберігання і деяку обробку даних (найчастіше – в базі даних чи іншому сховищі). Тут особливий інтерес представляє тестування даних, перевірка дотримання бізнес-правил, тестування продуктивності.

- Тестування чорної скриньки.

Це процес тестування системи або програмного забезпечення без попереднього знання того, як вона працює зсередини. Це стосується не лише незнання самого вихідного коду, але й того, що ви не бачили жодної проектної документації, пов'язаної з програмним забезпеченням. Тестувальники просто надають вхідні дані та отримують вихідні дані, як це робив би кінцевий користувач.

1.3 Димове тестування

Димове тестування - це процес тестування програмного забезпечення, який визначає, чи розгорнута збірка програмного забезпечення є стабільною чи ні. Димове тестування - це підтвердження для команди забезпечення якості (QA) для продовження подальшого тестування програмного забезпечення. Воно складається з мінімального набору тестів, що запускаються на кожній збірці для перевірки функціональності програмного забезпечення. Димове тестування також відоме як "Build Verification Testing" або "Confidence Testing".

Простіше кажучи, димові тести означають перевірку того, чи важливі функції працюють, і чи немає критичних помилок у збірці, що тестується. Це швидкий міні-регресивний тест основної функціональності. Це простий тест, який показує, що продукт готовий до тестування. Це допомагає визначити, чи збірка є дефектною, щоб подальше тестування не стало марною тратою часу та ресурсів.

Димове тестування проводиться щоразу, коли нові функціональні можливості програмного забезпечення розробляються та інтегруються з

існуючою збіркою, що розгортається в середовищі QA/проміжному середовищі. Це гарантує, що всі критичні функціональні можливості працюють правильно чи ні.

У цьому методі тестування команда розробників розгортає збірку в QA. Вибирається підмножина тестових випадків, а потім тестери запускають тестові випадки на збірці. Команда QA тестує додаток на критичні функціональні можливості. Ці серії тестових випадків призначені для виявлення помилок, які є в збірці. Якщо ці тести пройдені, команда QA продовжує функціональне тестування.

Будь-який збій означає необхідність повернути систему розробникам. За кожної зміни збірки ми проводимо димове тестування для забезпечення стабільності.

Димові тести дозволяють визначити придатність збірки для подальшого формального тестування. Головна мета димового тестування - виявити ранні серйозні проблеми. Димові тести призначені для демонстрації стабільності системи та відповідності вимогам. Збірка включає всі файли даних, бібліотеки, модулі повторного використання, інженерні компоненти, необхідні для реалізації однієї або кількох функцій продукту.

Якщо ми не проводимо димове тестування на ранніх стадіях, дефекти можуть бути виявлені на пізніших стадіях, де це може бути дорожче. І дефект, виявлений на пізніших стадіях, може стати критичним і вплинути на випуск поставок.

Після випуску збірки в середовище QA димове тестування виконується інженерами QA/керівником QA. За кожної нової збірки команда QA визначає основну функціональність у додатку для виконання димового тестування. Команда QA перевіряє наявність критичних помилок у додатку, що тестується.

Автоматизоване тестування використовується для регресійного тестування. Однак ми також можемо використовувати набір автоматизованих тестових кейсів для запуску димового тестування. За допомогою автоматизованих тестів розробники можуть перевірити збірку відразу після того, як нова збірка буде готова до розгортання.

Замість того, щоб повторювати тест вручну кожного разу, коли розгортається нова збірка програмного забезпечення, записані тестові кейси для димового тестування виконуються на збірці. Це дозволяє перевірити, чи основні функціональні можливості все ще працюють належним чином. Якщо тест не пройдено, то вони можуть виправити збірку та одразу розгорнути її повторно. Це дозволяє заощадити час і забезпечити якісну збірку в середовищі QA.

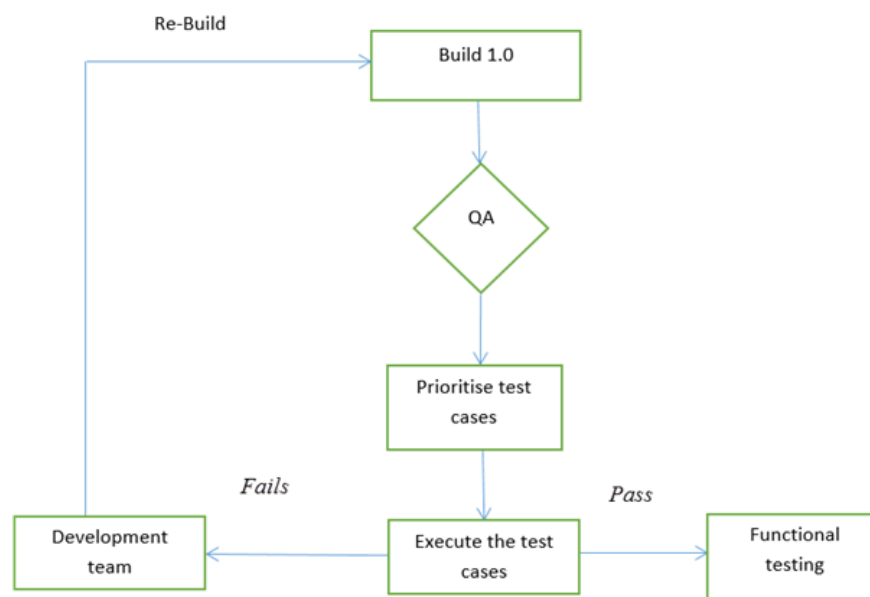


Рис. 1.3. Цикл димового тестування.

Переваги димового тестування:

- Простота виконання тестування
- Рання ідентифікація дефектів
- Підвищення якості системи

- Зниження ризиків
- Легкий доступ до прогресу
- Економія часу та зусиль на тестування
- Легке виявлення та виправлення критичних помилок
- Швидке виконання
- Мінімізація ризиків інтеграції

Недоліки димового тестування:

- Неохоплює всіх функцій програми
- Помилки можуть залишатися
- Знизька ефективність для великих проектів
- Негативні тести не використовуються
- Обмеження у виявленні проблем

1.4 Web-додатки

Web-додаток – клієнт-серверний додаток, в якому клієнт взаємодіє з вебсервером за допомогою браузера.

Програмне забезпечення веб-додатків складається з програм та даних, призначених для роботи в усьому світі, поєднання мережевого та клієнт-серверного програмного та апаратного забезпечення, виконуючи операції між локальними та віддаленими користувачами комп'ютера.

Клієнт-серверна архітектура – це архітектура, в якій мережеве навантаження розділяється між постачальниками послуг (серверами) і замовниками (клієнтами).[6]

Клієнт і сервер це програмне забезпечення розміщене на різних розрахункових машинах, що взаємодіє через мережеві протоколи. Можуть бути розміщені на одній машині (рис. 1.3).

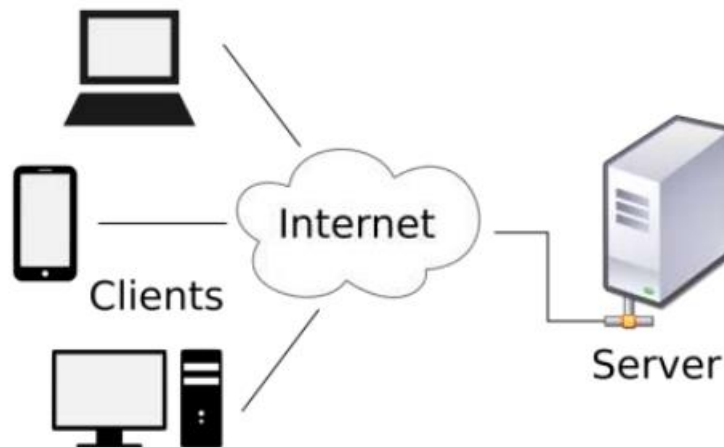


Рис. 1.3. Клієнт-серверна архітектура.

Переваги та недоліки клієнт-серверної архітектури:

- ✓ Відсутність дублювання коду програми сервера і клієнта.
- ✓ Всі дані на сервері і він більш захищений.
- ✓ Простіше контролювати доступ.
- Вартість
- Залежність від роботи сервера.

Запит посилає клієнт на сервер, а сервер реагує на запит (request) і віддає свій Http response (відповідь).

Запит складається з таких компонентів як:

- Метод;

Методи поділяються на (GET, POST, PUT, DELETE, CONNECT, OPTIONS, PATH)

- Версія протоколу;

Поділяються на HTTP/1.1, HTTP/2(Більш захищеніший)

- Хост машини;

Місце знаходження ресурсу. (URL)

- Хедер.

Додаткова інформація, яку містить відправлений ресурс (рис. 1.4).[7]

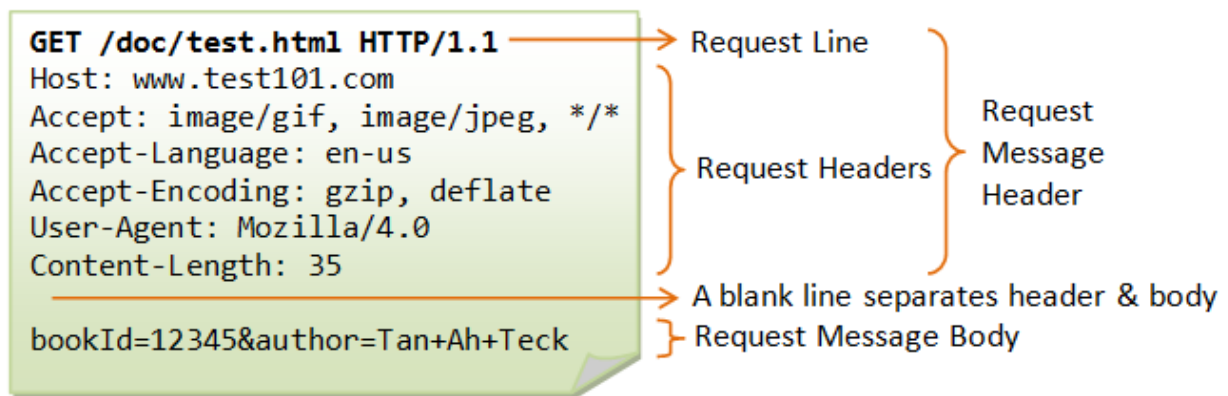


Рис. 1.4. Http request.

Відповідь на запит складається з таких компонентів як:

- Версія протоколу;

Поділяються на HTTP/1.1, HTTP/2 (більш захищеніший)

- Статус стану коду;

Вказує на успішність або неуспішність запиту.

Поділяється на:

1xx – переадресація, 100 – вказує, що браузер відправив запит на сервер, але процес відправки ще продовжується.

2xx – такі коди інформують про те, що сервер прийняв запит браузера і процес пройшов успішно.

3xx – з’являється, коли сервер вказує, що сторінка, яку ми запрошуємо, тимчасово або назавжди видалена та ресурс не доступний.

4xx – такі повідомлення свідчать про помилку у запиті.

5xx – такі повідомлення свідчать про помилку на сервері.

- Статус повідомлення;

Опис стану коду.

- Хедер.

Додаткова інформація, яку містить відправлений ресурс (рис. 1.5).[7]

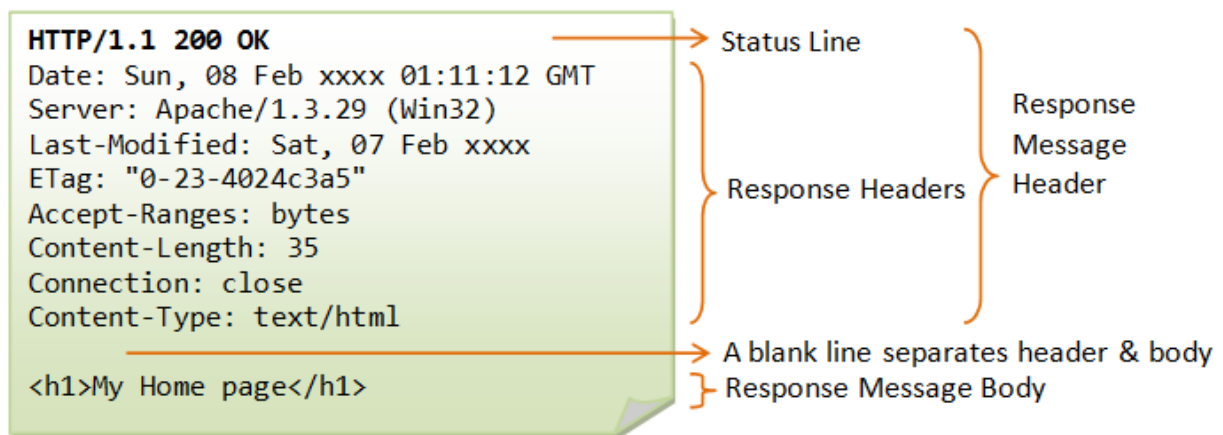


Рис. 1.5. Http response.

1.5 Характеристика інструменту Selenium

Selenium – це набір інструментів, призначених для автоматизації веббраузерів на різних платформах. Selenium може автоматизувати безліч різноманітних браузерів на різних платформах, використовуючи різні мови програмування і інтегруючись з різними тестовими фреймворками.

Selenium WebDriver – це гнучкий інструмент для автоматизованого тестування веб-проектів на базі набору бібліотек для різних мов програмування, таких як Java, .Net (C#), Python, Ruby, PHP, Perl, JavaScript. Даний інструмент підтримує роботу на базі Windows, macOS та Linux, а також найпоширеніші браузери Google Chrome, Firefox, Safari, Edge, Internet Explorer та навіть деякі браузери без графічного інтерфейсу. Використовується Selenium WebDriver найчастіше з такими видами тестування, як регресійне та функціональне.

Як і всі засоби автоматизації інтерфейсу користувача, Selenium дозволяє нам імітувати дії миші та клавіатури від імені користувача та отримувати дані, які відображаються.

Переваги:

- інтеграція з великою кількістю мов програмування та кросплатформність;
- простий набір команд та легкість створення скриптів за допомогою бібліотек;
- виконання авто-тестів можна здійснювати без участі людини та у будь-який час;
- безкоштовний продукт з відкритим вихідним кодом.

Недоліки:

- необхідність володіння навичками програмування;
- обмеженість функціоналу в порівнянні з платними аналогами;
- не може бути використаний для тестування графічних елементів та Flash-об'єктів;
- наявність дефектів у самих бібліотеках.

Архітектура Selenium WebDriver складається з 4-х компонентів:

- драйвер конкретного браузера;
- клієнтська бібліотека Selenium;
- браузер;
- протокол JSON Wire (JavaScript Object Notation).

Драйвер браузера отримує запити від мовної прив'язки та викликає файл відповідної операції в браузері. Кожен тип браузера має власний драйвер. Оскільки всі драйвери «розуміють» один і той же дротовий протокол JSON, той самий тест може використовуватися з іншим драйвером та відповідним браузером (рис. 2.1).



Рис. 2.1. Selenium WebDriver Automation Framework.

Selenium також пропонує спеціальний плагін для Chrome і Firefox, який називається Selenium-IDE. Цей плагін дозволяє записувати тестові кейси, а також базове управління та редагування цих тестових кейсів без написання коду. Це також дозволяє експортувати тести в код на Ruby, Java або C #, використовуючи різноманітні популярні фреймворки тестування.

2. РОЗРОБКА ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Опис Web-додатку

Для автоматизованого тестування було обрано веб-додаток “ToDo”. Веб-застосунок “ToDo” - це простий додаток, який дозволяє користувачам створювати та керувати списками завдань. Додаток має такі основні функції:

- Реєстрація користувачів
- Оновлення користувачів
- Видалення користувачів
- Призначення списків завдань користувачам
- Оновлення списків завдань
- Видалення списків завдань
- Додавання завдань до списків
- Оновлення завдань
- Видалення завдань з списків
- Додавання та видалення користувачів з списків

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

Планується три етапи проведення процесу тестування:

- перший етап полягає в складанні тест-кейсів;
- другий етап буде складатися з тестування за допомогою Selenium Webdriver;
- на третьому етапі буде проведений прогін авто тестів Selenium RC;

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

ОС, яка буде використовуватися під час тестування:

- Windows 11

Браузер, який буде використовуватися під час тестування:

- Firefox 121.0(64-bit)

2.2 Необхідні інструменти для створення автотестів

Selenium WebDriver – це простий API, який може допомогти в автоматизації браузера. Однак набагато більше необхідний при використанні його для тестування та побудови тестової основи.

Нам знадобиться інтегроване середовище розробки (IDE) або редактор коду для створення нового проєкту Java та додавання Selenium WebDriver та інші залежності для побудови основи тестування. Selenium WebDriver встановлюємо із офіційного джерела: <https://www.selenium.dev/>.

Selenium IDE — це інтегроване середовище розробки для тестів Selenium. Він реалізований як розширення Firefox і дозволяє записувати, редагувати та налагоджувати тести: <https://addons.mozilla.org/en-GB/firefox/addon/selenium-ide/>.

Також будемо використовувати IntelliJ IDEA. IntelliJ IDEA аналізує код, шукаючи зв'язки між символами у всіх файлах проєкту та мовах.

Використовуючи цю інформацію, він надає допомогу в глибокому кодуванні, швидку навігацію, розумний аналіз помилок: <https://www.jetbrains.com/idea/download/?section=windows>.

У світі Java Eclipse є широко використовуваною IDE, а також IntelliJ IDEA та NetBeans. Eclipse забезпечує багатofункціональне середовище для розробки тесту Selenium WebDriver. Поряд з Eclipse, Apache Maven надає підтримку для управління всім життєвим циклом тесту проєкту. Maven використовується для визначення структури проєкту, залежностей, побудови та управління тестами. Ми будемо використовувати Maven для побудови тестового середовища Selenium WebDriver. Ще однією важливою перевагою використання Maven є те, що можна отримати всі Selenium файли бібліотеки та їх залежності шляхом налаштування файлу pom.xml. Maven автоматично завантажує необхідні файли зі сховища під час створення проєкту.

Необхідно також завантажити Maven: <https://maven.apache.org/>.

2.3 Вибір мови програмування

При виборі мови сценаріїв слід враховувати наступні фактори.

- Сучасна мова, що використовується для розвитку.
- Підтримка розробників мови програмування локально. Тестери повинні розглянути можливість використання мови, за якою стежать розробники.
- Для тих, хто новачок у процесі, краще використовувати Python, Ruby або будь-які інші мови, зручні для виконання сценаріїв. Він вимагає менше коду і може бути написаний швидко, без зайвих клопотів.

- Хоча Java є найпоширенішою мовою, її синтаксис дещо складний для реалізації в скриптах.

Ми реалізуємо наші автотести за допомогою Java. Java – це мова програмування загального призначення, яка належить корпорації Oracle. Java побудована на принципах об'єктно-орієнтованого програмування. Мова дотримується принципу WORA, який приносить багато переваг між платформами.

Багато великих корпорацій використовують Java для обслуговування своїх внутрішніх систем. Існує більше 3 мільярдів пристроїв, на яких запущено додатки, побудовані за допомогою Java.

Selenium є корисним інструментом, оскільки він є не лише відкритим кодом, а й портативною програмою тестування програмного забезпечення для вебдодатків, що підтримують різні мови, такі як Java, C #, Ruby, Python. Вибір правильної мови залежить від програми, що тестується, спільноти, що підтримує, доступних систем автоматизації тестів, зручності використання, елегантності та, звичайно, безперебійної інтеграції збірки. [18]

<https://www.oracle.com/java/>

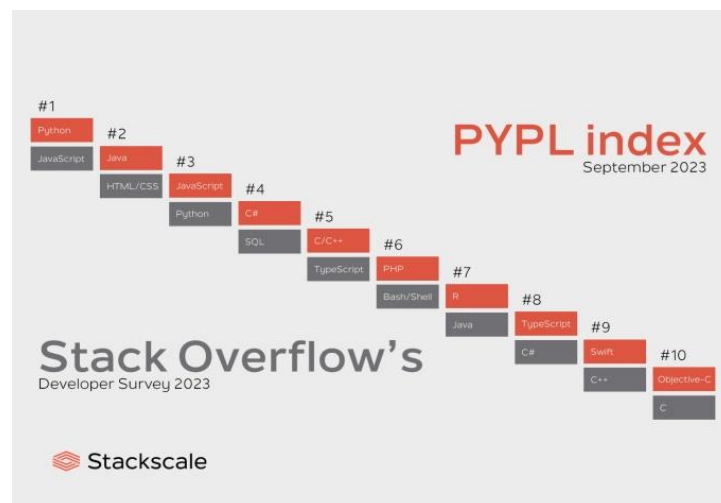


Рис. 2.1. Рейтинг найбільш популярних мов програмування.

2.4 Створення документації

Після встановлення усіх необхідних додатків, ми можемо переходити до опису документації. Перший етап полягає в складанні тест-кейсів. Це було зазначено вище в етапах, тому почнемо з цього.

Тест-кейс — це набір вхідних значень, попередніх умов виконання, очікуваних результатів і післяумов, розроблених для конкретної мети або умови тестування. Наприклад, для виконання певного програмного шляху або для перевірки відповідності певній вимозі.

- дозволяють зрозуміти покриття вимог вашими тестами;
- допомагають не забути, що саме ви протестували;
- систематизують всю подальшу роботу (якщо у вас виникнуть питання, чи тестували ви щось, то у вас буде підтвердження цього).

Під тест-кейсом також може розумітися відповідний документ, який представляє формальний запис тест-кейса (рис. 2.2.).

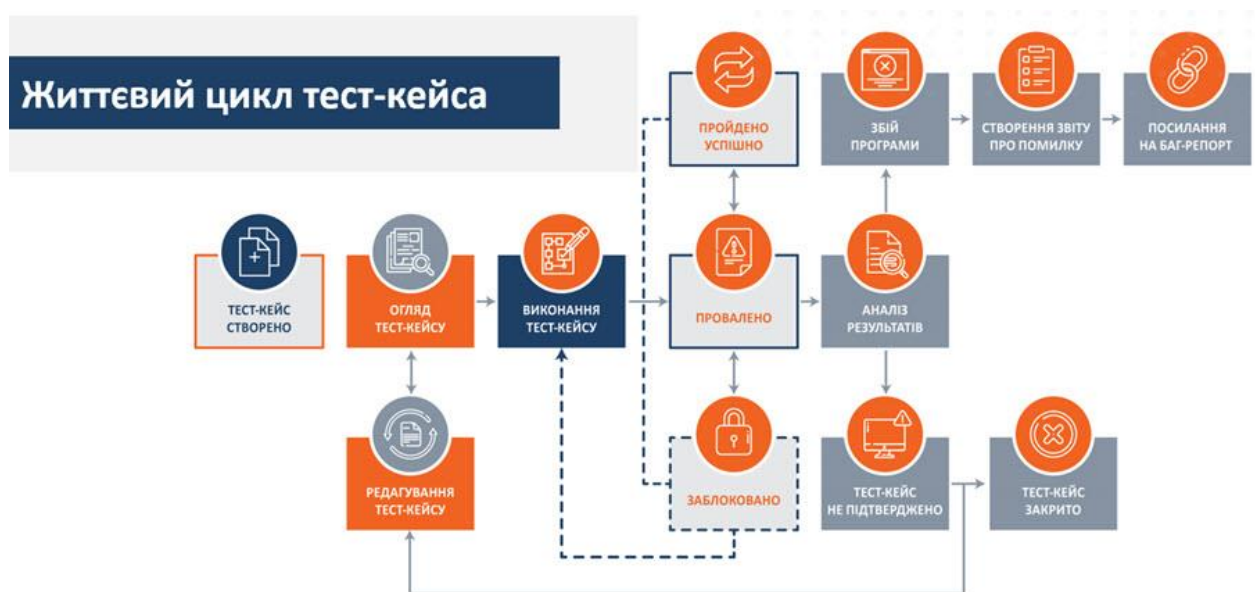


Рис. 2.2. Життєвий цикл тест-кейсу.

3. КІНЦЕВІ РЕЗУЛЬТАТИ

1.1 Автоматизоване димне тестування системи веб-додатку

3.1.1. Створення проєкту

У додатку IntelliJ IDEA створюємо новий проєкт за допомогою Maven і обираємо версію Java. Проєкт створили під назвою «TestingToDo» (рис. 3.1.).

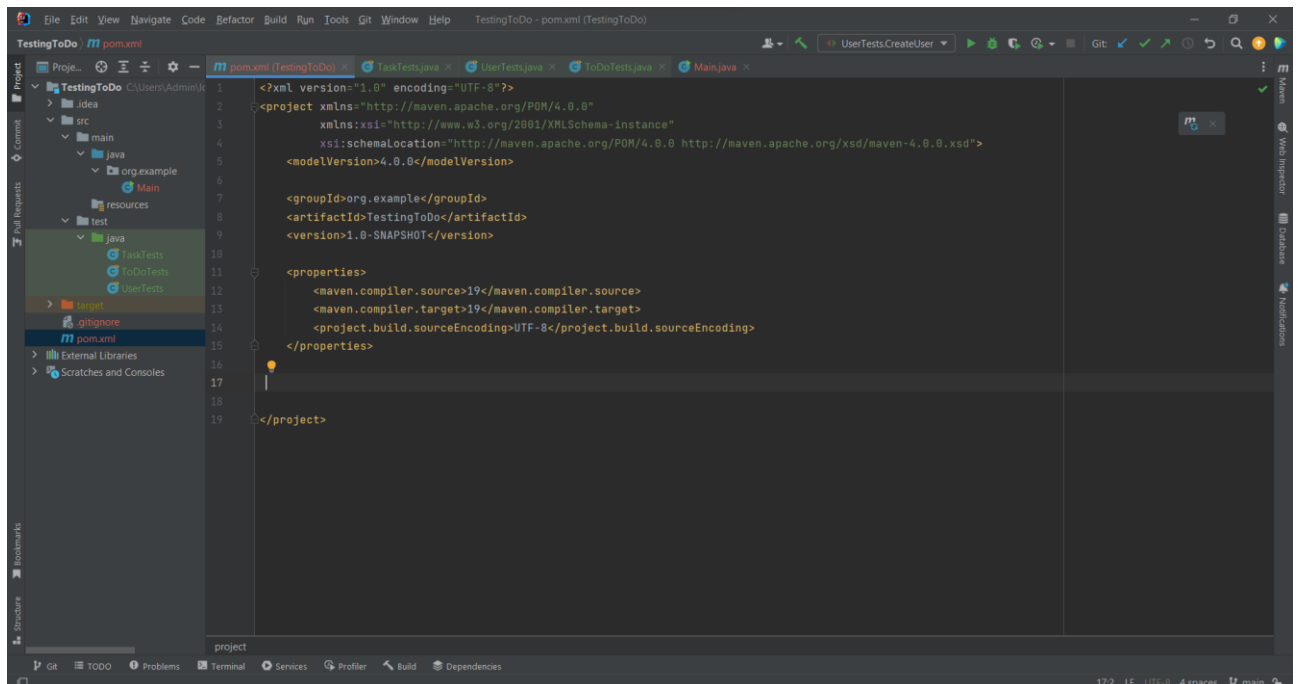


Рис. 3.1. Головний екран проєкту IntelliJ IDEA

Спочатку нам потрібно встановити прив'язки Selenium для проєкту автоматизації.

Встановлення бібліотек Selenium для Java можна здійснити за допомогою Maven.

<dependency>

<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-java</artifactId>


```

    <version>LATEST</version>

</dependency>

```

Щоб створити екземпляр сеансу Firefox, ми зробимо наступне:

```

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.Select;

public class UserTests {
    WebDriver driver;
    @Before
    public void setUp() {
        System.setProperty("webdriver.firefox.driver", "C:\\\\Program
Files\\\\Mozilla Firefox\\\\firefox.exe");
        //слід встановити залежність, що визначає шлях до firefoxdriver
        driver = new FirefoxDriver();
        //для запуску браузера необхідно створити об'єкт драйвера
    }

```

Щоб завершити тест, викличемо quit метод на екземплярі WebDriver інтерфейсу, наприклад, на driver змінній.

```

    @After
    public void tearDown() {
        driver.quit();
    }

```

Також використаємо методи відкриття нашої сторінки на повний екран.

```

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

```

Створимо проект Selenium IDE “Testing ToDo”

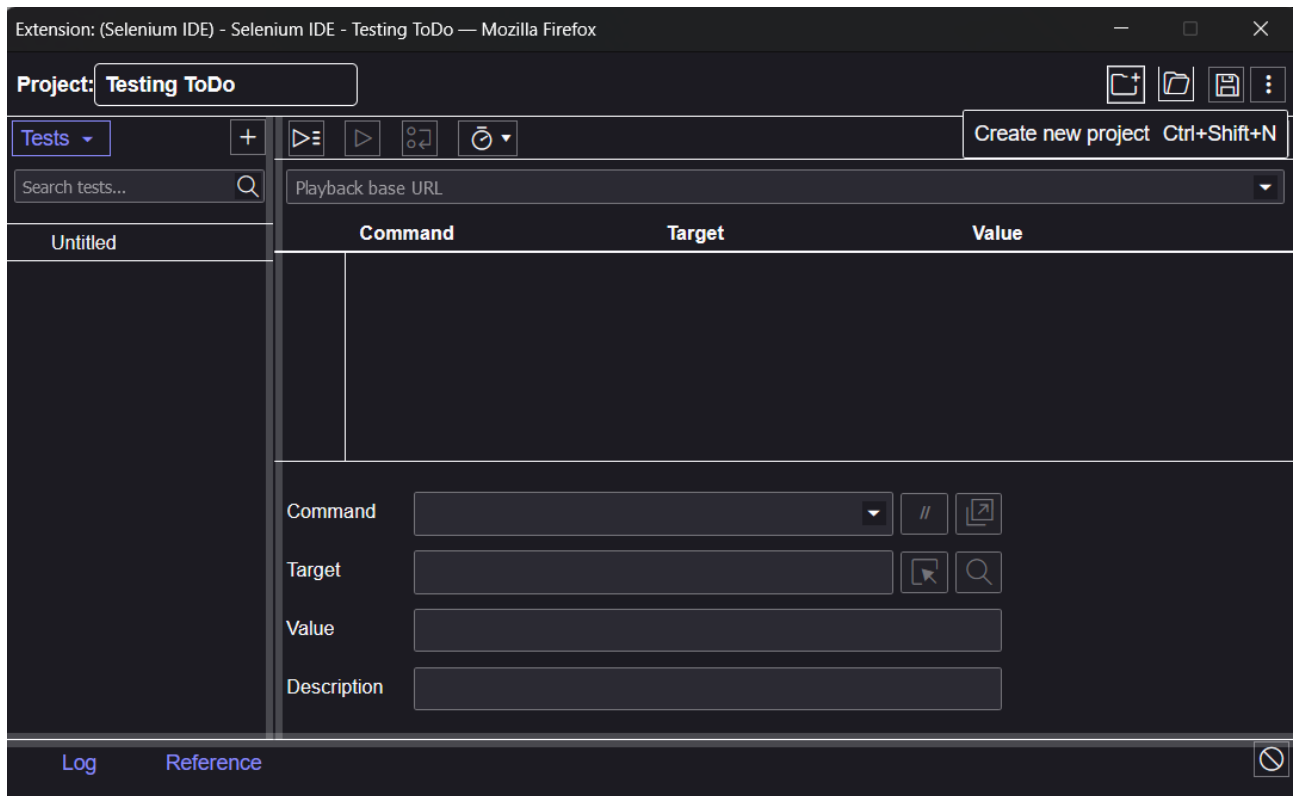


Рис. 3.2. Головний екран проєкту Selenium IDE.

На цьому налаштування нашого проєкту автоматизації завершено. Далі переходимо до написання самих тест-кейсів та створення автотестів до них.

3.1.2. Створення тест-кейсів

Для нашого застосунку створимо декілька тест-кейсів з описом кроків тестування для автотестів. Це будуть тільки деякі найбільш використовувані кроки. Розглянемо лише позитивні сценарії, адже проводимо димове тестування.

Тест-кейс №1: Створення користувача

Опис: Перевірити, що користувач може бути успішно створений з

правильними даними.

Кроки:

1. Перейти на сторінку створення користувача (<http://localhost:8081/users/create>).
2. Заповнити всі необхідні поля (ім'я, прізвище, email, пароль).
3. Натиснути кнопку "Зареєструватися".
4. Перевірити, що користувач з'явився в списку користувачів (<http://localhost:8081/>).

Критерії успіху:

- Користувач повинен бути створений і відображатися в списку користувачів.

4	1	Alexandra Kyrylchuk	alex@mail.com	Edit	Remove
---	---	---------------------	---------------	------	--------

Рис. 3.3. Скріншот після виконання автотесту.

4.	click on id=first-name OK	18:59:18
5.	type on id=first-name with value Oleksandra OK	18:59:18
6.	click on id=last-name OK	18:59:19
7.	type on id=last-name with value Kyrylchuk OK	18:59:19
8.	click on id=email OK	18:59:19
9.	type on id=email with value alex@mail.com OK	18:59:19
10.	click on id=password OK	18:59:19
11.	type on id=password with value 1111 OK	18:59:19
12.	click on id=register OK	18:59:19
13.	click on css=html OK	18:59:20
14.	click on linkText=TODOs List OK	18:59:20
'User Create' completed successfully		18:59:20

Рис. 3.4. Результат перевірки тесту в Selenium IDE.

Тест-кейс №2: Оновлення користувача

Опис: Перевірити, що інформація про користувача може бути успішно оновлена.

Кроки:

1. Перейти на сторінку редагування користувача (<http://localhost:8081/users/6/update>).
2. Внести зміни в необхідні поля (ім'я, прізвище, пароль, роль).
3. Натиснути кнопку "Оновити".
4. Перевірити, що зміни вступили в силу в списку користувачів (<http://localhost:8081/>).

Критерії успіху:

- Інформація про користувача повинна бути оновлена в списку користувачів.

4	6	Noraedit Whiteedit	nora@mail.com	Edit	Remove
---	---	--------------------	---------------	------	--------

Рис. 3.5. Скріншот після виконання автотесту.

4.	click on id=first-name OK	19:04:08
5.	type on id=first-name with value Noraedit OK	19:04:09
6.	click on id=last-name OK	19:04:09
7.	type on id=last-name with value Whiteedit OK	19:04:09
8.	click on id=old-password OK	19:04:09
9.	type on id=old-password with value 3333 OK	19:04:10
10.	click on id=new-password OK	19:04:10
11.	type on id=new-password with value 3333 OK	19:04:10
12.	click on id=update OK	19:04:10
13.	click on css=html OK	19:04:10
14.	click on linkText=TODOs List OK	19:04:11
'User Update' completed successfully		19:04:11

Рис. 3.6. Результат перевірки тесту в Selenium IDE.

Тест-кейс №3: Видалення користувача

Опис: Перевірити, що користувач може бути успішно видалений.

Кроки:

1. Перейти на сторінку списку користувачів (<http://localhost:8081/>).

2. Натиснути кнопку "Створити" -> перехід на іншу сторінку.
3. Заповнити поле "Назва".
4. Натиснути кнопку "Створити".
5. Перевірити, що завдання з'явилося в списку завдань.

Критерії успіху:

- Список завдання повинно бути створено і відображатися в списку завдань користувача.

6	1	New TODO	30.12.2023 20:58	Mike Brown	Edit	Remove
---	---	----------	------------------	------------	------	--------

Рис. 3.10. Скріншот після виконання автотесту.

Running 'ToDo Create'	18:54:16
1. open on / OK	18:54:16
2. setWindowSize on 880x816 OK	18:54:16
3. click on linkText=Nora White OK	18:54:16
4. click on id=create OK	18:54:17
5. click on id=title OK	18:54:17
6. type on id=title with value New ToDo List OK	18:54:17
7. click on id=create OK	18:54:17
'ToDo Create' completed successfully	18:54:18

Рис. 3.11. Результат перевірки тесту в Selenium IDE.

Тест-кейс №5: Оновлення списку завдань

Опис: Перевірити, що інформація про список завдання може бути успішно оновлена.

Кроки:

1. Перейти на сторінку редагування завдання (<http://localhost:8081/todos/9/update/users/5>).
2. Внести зміни в поле "Назва".
3. Натиснути кнопку "Оновити".
4. Перевірити, що зміни вступили в силу в списку завдань.

Критерії успіху:

- Інформація про завдання повинна бути оновлена в списку завдань.

1	9	Mike's To-Do #3 Edited	16.09.2020 14:00	Mike Brown	Edit	Remove
---	---	------------------------	------------------	------------	------	--------

Рис. 3.12. Скріншот після виконання автотесту.

Running 'ToDo Update'	18:57:04
1. open on / OK	18:57:04
2. setWindowSize on 880x816 OK	18:57:04
3. click on linkText=Nora White OK	18:57:04
4. click on linkText=Edit OK	18:57:05
5. click on id=title OK	18:57:05
6. type on id=title with value Nick's To-Do #1 Edited OK	18:57:05
7. click on id=update OK	18:57:05
'ToDo Update' completed successfully	18:57:06

Рис. 3.13. Результат перевірки тесту в Selenium IDE

Тест-кейс №6: Видалення списку завдань

Опис: Перевірити, що завдання може бути успішно видалено.

Кроки:

1. Перейти на сторінку списку завдань (<http://localhost:8081/todos/all/users/5>).
2. Натиснути кнопку "Видалити" біля потрібного списку.
3. Підтвердити видалення.
4. Перевірити, що завдання зникло зі списку завдань.

Критерії успіху:

- Завдання повинно бути видалено зі списку завдань.

No.	Id	Title	Created At	Owner	Operations	
1	12	Nora's To-Do #1	16.09.2020 14:15	Nora White	Edit	Remove
2	8	Mike's To-Do #2	16.09.2020 14:00	Mike Brown	Edit	Remove
3	7	Mike's To-Do #1	16.09.2020 14:00	Mike Brown	Edit	Remove
4	11	Nick's To-Do #2	16.09.2020 14:15	Nick Green	Edit	Remove

Рис. 3.14. Скріншот після виконання автотесту.

- Завдання повинно бути створено і відображатися в списку завдань.
- Співавтор повинен бути доданий до завдання та відображатися у списку співавторів.

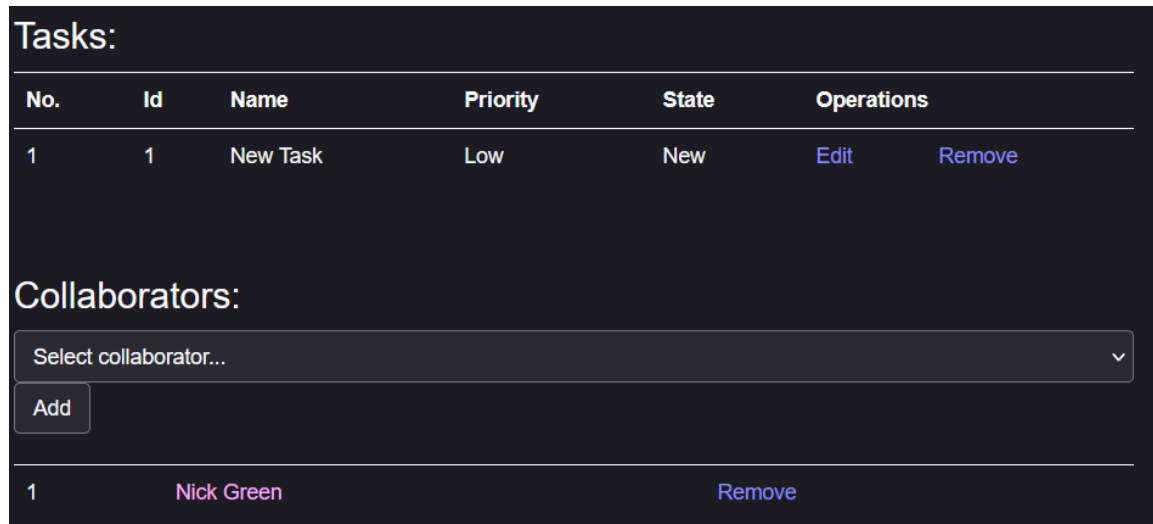


Рис. 3.17. Скріншот після виконання автотесту.



Рис. 3.18. Результат перевірки тесту в Selenium IDE.

Тест-кейс №8: Оновлення завдання

Опис: Перевірити, що інформація про завдання (назва, пріоритет, статус) може бути успішно оновлена.

Кроки:

1. Перейти на сторінку редагування завдання (<http://localhost:8081/tasks/7/update/todos/7>).

2. Внести зміни в поле "Назва".
3. Вибрати пріоритет зі спадного списку.
4. Вибрати статус зі спадного списку.
5. Натиснути кнопку "Оновити".
6. Перевірити, що зміни вступили в силу в списку завдань.

Критерії успіху:

- Інформація про завдання повинна бути оновлена в списку завдань.

2	7	Task #3 Edited	High	Done	Edit	Remove
---	---	----------------	------	------	------	--------

Рис. 3.19. Скріншот після виконання автотесту.

4.	click on linkText=Mike's To-Do #1 OK	18:47:34
5.	click on linkText=Edit OK	18:47:34
6.	click on id=name OK	18:47:34
7.	type on id=name with value Task #2 Edited OK	18:47:34
8.	click on id=priority OK	18:47:35
9.	select on id=priority with value label=Medium OK	18:47:35
10.	click on css=#priority > option:nth-child(2) OK	18:47:35
11.	click on id=state OK	18:47:35
12.	select on id=state with value label=Doing OK	18:47:35
13.	click on css=#state > option:nth-child(2) OK	18:47:35
14.	click on id=update OK	18:47:35
'Task Update' completed successfully		18:47:36

Рис. 3.20. Результат перевірки тесту в Selenium IDE.

Тест-кейс №9: Видалення завдання

Опис: Перевірити, що завдання може бути успішно видалено.

Кроки:

1. Перейти на сторінку списку завдань Todo (<http://localhost:8081/todos/7/tasks>).
2. Натиснути кнопку "Видалити" біля потрібного завдання.
3. Підтвердити видалення.
4. Перевірити, що завдання зникло зі списку завдань.

Критерії успіху:

- Завдання повинно бути видалено зі списку завдань.

Tasks:

No.	Id	Name	Priority	State	Operations	
1	5	Task #1	High	Done	Edit	Remove
2	7	Task #3 Edited	High	Done	Edit	Remove

Рис. 3.21. Скріншот після виконання автотесту.

Running 'Task Delete'	18:49:54
1. open on / OK	18:49:55
2. setWindowSize on 880x693 OK	18:49:55
3. click on linkText=Nick Green OK	18:49:55
4. click on css=html OK	18:49:56
5. click on css=tr:nth-child(3) > td:nth-child(7) > a OK	18:49:56
6. click on linkText=Mike's To-Do #1 OK	18:49:56
'Task Delete' completed successfully	18:49:57

Рис. 3.22. Результат перевірки тесту в Selenium IDE.

Run: TaskTests	Tests passed: 3 of 3 tests – 26 sec 342 ms	
TaskTests	26 sec 342 ms	"C:\Program Files\Java\jdk-19\bin\java.exe" ...
CreateTask	12 sec 452 ms	Process finished with exit code 0
DeleteTask	6 sec 721 ms	
UpdateTask	7 sec 169 ms	

Рис. 3.23. Результат перевірки тесту в Selenium IntelliJ IDEA.

1.2 Аналіз отриманих результатів

Усі тест-кейси, які були розроблені для проведення димового тестування, пройшли успішно. Це означає, що основні функції веб-додатку працюють належним чином.

Особливо варто відзначити, що тести перевірили такі важливі функції, як:

- Створення, оновлення та видалення користувачів;
- Створення, оновлення та видалення списків завдань;
- Створення, оновлення та видалення завдань.

Таким чином, можна зробити висновок, що веб-додаток готовий до подальшого тестування. Однак, щоб мати більш повне уявлення про стан додатку, рекомендується провести додаткові тести, які б перевірили інші функції та сценарії використання.

Selenium IDE - це плагін для браузера Firefox, який дозволяє записувати і відтворювати сценарії тестування. Він є найпростішим інструментом у сімействі Selenium, але його можливості обмежені. Selenium IDE підтримує лише один браузер, Firefox, і лише одну мову програмування, Selenese.

Selenium WebDriver - це найновітніший і найпотужніший інструмент у сімействі Selenium. Він підтримує всі основні браузери і всі основні мови програмування. Selenium WebDriver також є найбільш гнучким інструментом, оскільки дозволяє розробникам писати сценарії тестування на основі об'єктно-орієнтованого програмування.

Selenium IDE є хорошим вибором для початківців, які хочуть швидко і легко розпочати автоматизацію тестування веб-додатків. Selenium WebDriver - це найкращий вибір для досвідчених розробників, які потребують найбільш гнучкого і потужного інструменту для автоматизації тестування веб-додатків.

ВИСНОВКИ

У курсовій роботі було розглянуто теоретичні основи димового тестування веб-додатків, а також розроблено систему автотестів для веб-застосунку “ToDo” за допомогою інструменту Selenium WebDriver та Selenium IDE.

На основі проведеного дослідження можна зробити наступні висновки:

- Димове тестування є важливим етапом тестування програмного забезпечення, який дозволяє визначити, чи готова збірка до подальшого тестування.
- Автоматизоване димове тестування дозволяє виконувати тестування швидко та ефективно, що особливо важливо для великих і складних проектів.
- Selenium WebDriver є потужним і гнучким інструментом, який дозволяє автоматизувати тестування веб-додатків на різних платформах за допомогою різних мов програмування.

У процесі роботи над курсовою роботою було досягнуто поставлених цілей та виконано всі поставлені завдання. Було розроблено систему автотестів для веб-застосунку “ToDo”, яка дозволяє перевірити основні функціональні можливості додатку. Тести були написані на мові програмування Java та використовують інструменти Selenium WebDriver та Selenium IDE.

У подальшій роботі над курсовою роботою можна було б додати наступні можливості:

- Розширити набір тестів для перевірки додаткових функціональних можливостей веб-застосунку.
- Додати можливість автоматичного запуску тестів за розкладом.
- Забезпечити інтеграцію системи автотестів з іншими системами управління тестуванням.

На основі проведеного дослідження можна дати наступні рекомендації для проведення димового тестування веб-додатків:

- Використовуйте автоматизоване димове тестування для великих і складних проектів.
- Використовуйте інструменти, які підтримують широкий діапазон браузерів і мов програмування.
- Створюйте систему автотестів, яка дозволяє перевірити основні функціональні можливості веб-застосунку.
- Регулярно запускайте димове тестування для виявлення помилок на ранніх етапах розробки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Automation Testing Tutorial: What is Automated Testing. [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.guru99.com/automation-testing.html>
2. Клієнт-серверна архітектура [Електронний ресурс] – Режим доступу до ресурсу:
https://uk.wikipedia.org/wiki/%D0%9A%D0%BB%D1%96%D1%94%D0%BD%D1%82-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80%D0%BD%D0%B0_%D0%B0%D1%80%D1%85%D1%96%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0
3. Обзор протокола HTTP [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.mozilla.org/ru/docs/Web/HTTP/Overview>
4. Тестування програмного забезпечення [Електронний ресурс] – Режим доступу до ресурсу:
https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F
5. An all-in-one test automation solution [Електронний ресурс] – Режим доступу до ресурсу: <https://www.katalon.com/>
6. Complete Guide to Test Automation: Techniques, Practices, and Patterns for Building and Maintaining Effective Software Projects [Електронний ресурс] Режим доступу до ресурсу:
<http://tisten.ir/wp-content/uploads/2019/01/Complete-Guide-to-TestAutomation-Techniques-Practices-and-Patterns-for-Building->

[andMaintaining-Effective-Software-Projects-Apress-2018-Arnon-Axelrod.pdf](#)

7. Driver requirements Maven [Электронный ресурс] – Режим доступа до ресурсу:
https://www.selenium.dev/documentation/en/webdriver/driver_requirements/
8. Getting Started with Selenium for Automated Website Testing [Электронный ресурс] – Режим доступа до ресурсу:
<https://wiki.saucelabs.com/display/DOCS/Getting+Started+with+Selenium+for+Automated+Website+Testing>
9. Installing Selenium libraries [Электронный ресурс] – Режим доступа до ресурсу:
https://www.selenium.dev/documentation/en/selenium_installation/installing_selenium_libraries/
10. IntelliJ IDEA [Электронный ресурс] – Режим доступа до ресурсу:
<https://www.jetbrains.com/ru-ru/idea/>
11. Oracle Java [Электронный ресурс] – Режим доступа до ресурсу:
<https://www.oracle.com/java/>
12. Selenium Webdriver [Электронный ресурс] – Режим доступа до ресурсу:
https://drive.google.com/file/d/0B7TBmsv_w76nQ1FfQVdyWWRXYk0/view
13. Selenium with Java : Getting Started to Run Automated Tests [Электронный ресурс] – Режим доступа до ресурсу:
<https://www.browserstack.com/guide/selenium-with-java-for-automated-test>
14. Strategies for Testing Web Applications from the Client Side [Электронный ресурс] – Режим доступа до ресурсу:
<https://cs.fit.edu/media/TechnicalReports/cs-2003-11.pdf>
15. Welcome to Apache Maven [Электронный ресурс] – Режим доступа до ресурсу: <https://maven.apache.org/>

16. What is Smoke Testing? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.guru99.com/smoke-testing.html>
17. Тест-кейси: що, як, навіщо. [Електронний ресурс] – Режим доступу до ресурсу: https://dou.ua/forums/topic/46769/?from=comment-digest_post&utm_source=digest-comments&utm_medium=email&utm_campaign=25122023