

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ
ПРОВЕДЕННЯ ДИМНОГО ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ
SOFTWARE DEVELOPMENT FOR CONDUCTING A SMOKE
TEST OF A WEB APPLICATION

Виконав(ла): студент(ка) 4 курсу, групи КНТ-130
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

АКОПЯН Г.Р.

(ПРИЗВИЩЕ та ініціали)

Керівник ГЛАДКОВА О.М.

(ПРИЗВИЩЕ та ініціали)

Рецензент ЗЕЛЕНЬОВА І.Я.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2024 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

АКОПЯНА Гарегіна Рафіковича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка програмного забезпечення для проведення димного тестування вебзастосунку. Software Development for Conducting a Smoke Test of a Web Application.

керівник проєкту (роботи) к.т.н., ГЛАДКОВА Ольга Миколаївна,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 24 ” квітня 2024 року № 168

2. Строк подання студентом проєкту (роботи) 10 червня 2024 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Основні концепції тестування. 2. Огляд та вибір технологій та інструментів розробки. 3. Розробка програмного забезпечення димного тестування вебзастосунку. 4. Інструкції по запуску автотестів.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ГЛАДКОВА О.М., доцент		
Нормоконтроль	БЄЛОВА А.В., асистент		

7. Дата видачі завдання “ 15 ” квітня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис)

Гарегін АКОПЯН

(Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

(підпис)

Ольга ГЛАДКОВА

(Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 61 с., 4 табл., 23 рис., 2 дод., 17 джерел.

АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ, ДИМНЕ ТЕСТУВАННЯ, SELENIUM WEBDRIVER, JAVA, INTELLIJ IDEA.

Об'єкт дослідження – програмні засоби для автоматизації тестування вебзастосунків.

Предмет дослідження – програмне забезпечення для автоматизованого димного тестування вебзастосунку.

Мета роботи – розробка програмного забезпечення для автоматизації димного тестування сайту, що дозволить швидко та ефективно перевіряти основну працездатність сайту після кожного збирання коду.

Матеріали, методи та технічні засоби: Мови програмування Java, технологія та інструменти Selenium WebDriver, TestNG, середовище розробки IntelliJ IDEA, об'єктно-орієнтоване програмування, паттерн Page Object Model (POM), Jenkins для автоматичного запуску тестів.

Результати. Створено програмне забезпечення, яке автоматизує процес димного тестування вебзастосунку. Програмне забезпечення дозволяє швидко виконувати тестові сценарії для перевірки основної функціональності сайту, включаючи реєстрацію, вхід користувача, створення та відправлення листівок, вибір та покупку подарунків, а також здійснення платежів.

Висновки. Розроблено програмне забезпечення для автоматизованого димного тестування сайту, яка дозволяє прискорити та підвищити ефективність процесу тестування.

Галузь використання – в компаніях, що займаються розробкою вебзастосунків.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 61 pages, 4 tables, 23 figures, 2 appendixes, 17 sources.

AUTOMATED TESTING, SMOKE TESTING, SELENIUM WEBDRIVER, JAVA, INTELLIJ IDEA.

Object of study is a software tools for automating the testing of web applications.

Subject of study are software for automated smoke testing of a web application.

The purpose of the work is to develop software for automating smoke testing of the site, which will allow you to quickly and efficiently check the basic functionality of the site after each code build.

Materials, methods, and tools: Java programming languages, Selenium WebDriver technology and tools, TestNG, IntelliJ IDEA development environment, object-oriented programming, Page Object Model (POM), Jenkins for automatic test launch.

Results. Created software that automates the process of smoke testing a web application. The software allows you to quickly run test scenarios to test the main functions of the site, including registration, user login, creating and sending postcards, selecting and purchasing gifts and making payments.

Conclusions. A software system for automated fumigation of the site has been developed, which allows you to speed up and increase the efficiency of the testing process.

The field of use is a companies engaged in the development of web applications.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	8
Вступ.....	9
1 Основні концепції тестування	11
1.1. Визначення тестування та його цілей	11
1.2 Види тестування	12
1.2.1 Функціональне тестування.....	12
1.2.2 Нефункціональне тестування	13
1.2.3. Автоматизоване тестування	15
1.2.4. Ручне тестування.....	16
1.2.5 Димне (смоук) тестування.....	18
1.3 Переваги та недоліки видів тестування	18
1.4 Висновки за розділом	20
2 Огляд та вибір технологій та інструментів розробки.....	21
2.1. Мова програмування.....	21
2.1.1 Переваги Java.....	21
2.1.2 Переваги Python	21
2.1.3 Порівняння мов програмування для створення автотестів	22
2.2 Вибір допоміжного інтерфейса для написання автотестів	23
2.2.1 Інструмент Selenium	23
2.2.2 Cypress.....	24
2.2.3 Порівняння Selenium та Cypress	25
2.3. Вибір середовища розробки.....	26
2.3.1 Середовище розробки IntelliJ IDEA.....	26
2.3.2 Порівняння IntelliJ IDEA з середовищем розробки Eclipse.....	28
2.4 Висновки за розділом	30
3 Розробка програмного забезпечення димного тестування вебзастосунку... 31	
3.1. Визначення тестових випадків	32

3.2. Реалізація тестових випадків	33
3.3 Структура автотестів	33
3.4 Опис автотестів	34
3.5 Алгоритми роботи автотестів	35
3.6 Висновки до розділу	39
4 Інструкції по запуску автотестів.....	40
4.1 Налаштування середовища розробки.....	40
4.2 Створення нового проєкту в IntelliJ IDEA.....	41
4.3 Налаштування проєкту	41
4.4 Реалізація та запуск тестових випадків.....	42
4.5 Висновки за розділом	44
Висновки	45
Перелік джерел посилання	46
Додаток А Текст програми.....	48
Додаток Б Слайди презентації	56

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

CI/CD – Continuous Integration/Continuous Deployment;

IDE – Integrated Development Environment;

JDK – Java Development Kit;

ООП – об'єктно орієнтоване програмування;

ПЗ – програмне забезпечення.

ВСТУП

Тестування є ключовим елементом процесу розробки програмного забезпечення, що спрямований на виявлення дефектів та забезпечення відповідності продукту визначеним вимогам. Воно є важливою складовою забезпечення якості, яка охоплює перевірку правильності функціонування системи, її продуктивності, безпеки та зручності використання. В умовах зростаючої складності програмного забезпечення та високих вимог до якості, тестування набуває все більшої важливості.

Існує багато видів тестування, кожен з яких має свої переваги та недоліки. Функціональне тестування забезпечує відповідність функціональним вимогам, нефункціональне – перевіряє такі аспекти, як продуктивність і безпека, автоматизоване тестування підвищує ефективність та точність, а ручне тестування дозволяє гнучко перевіряти нові функції.

Однак, особливу увагу заслуговує димне тестування, яке є пріоритетним методом перевірки на початкових етапах розробки. Димне тестування (або смоук тестування від англ. "smoke testing") отримало свою назву від аналогії з тестуванням електронних пристроїв, коли після вмикання перевіряють, чи не йде дим, що сигналізує про серйозні проблеми. У контексті програмного забезпечення, смоук-тестування – це поверхнева перевірка основної функціональності системи після кожного збирання (build), яка допомагає виявити критичні дефекти на ранніх стадіях.

Пріоритетність димного тестування обумовлена кількома важливими факторами: швидкість та ефективність, раннє виявлення критичних дефектів, регулярне виконання смоук-тестів підвищує загальну якість програмного забезпечення.

Таким чином, димне тестування виступає як перший бар'єр на шляху дефектів, що забезпечує швидку перевірку основної функціональності та стабільності системи. Враховуючи його ефективність та важливість, димне тестування займає пріоритетне місце у процесі забезпечення якості

програмного забезпечення. Таким чином тема дипломної кваліфікаційної роботи є актуальною.

Для досягнення поставленої мети були вирішені наступні завдання:

- аналіз існуючих видів тестування;
- аналіз теоретичних аспектів смюк тестування;
- розробка архітектури автоматизованої системи димного тестування;
- розробка тестових сценаріїв;
- реалізація автотестів.

1 ОСНОВНІ КОНЦЕПЦІЇ ТЕСТУВАННЯ

1.1. Визначення тестування та його цілей

Тестування – це процес перевірки програмного забезпечення або системи з метою виявлення дефектів та підтвердження відповідності технічним і функціональним вимогам. Основна мета тестування полягає у забезпеченні якості продукту, зниженні ризиків та підвищенні задоволення користувачів [1].

У процесі тестування беруть участь кілька ключових ролей. Тестувальник – виконує тестування та документує результати. Розробник створює та виправляє код програми. Менеджер з якості – координує процеси забезпечення якості.

Етапи тестування [1]:

- планування – визначення мети тестування, створення тестового плану;
- підготовка тестових сценаріїв – створення детальних описів тестових випадків;
- виконання тестів – запуск тестів і спостереження за результатами;
- документування результатів – запис результатів і звітність про дефекти;
- аналіз результатів – аналіз отриманих даних для покращення якості програмного забезпечення.

Цілі тестування [1]:

- виявлення дефектів – зниження кількості помилок у програмному забезпеченні;
- перевірка відповідності вимогам – підтвердження, що система відповідає специфікаціям;
- покращення якості – підвищення стабільності, продуктивності та безпеки програмного забезпечення;

– задоволення користувачів – забезпечення позитивного досвіду користувачів при взаємодії з системою.

1.2 Види тестування

1.2.1 Функціональне тестування

Цей вид тестування перевіряє, чи система виконує свої функції відповідно до вимог. Методи функціонального тестування включають [2]:

- модульне тестування – перевірка окремих компонентів;
- інтеграційне тестування – тестування взаємодії між компонентами;
- системне тестування – перевірка всієї системи в цілому.

Функціональне тестування є одним із найважливіших видів тестування, оскільки воно перевіряє, чи виконує система свої функції відповідно до визначених вимог. Основною метою функціонального тестування є виявлення дефектів, пов'язаних з конкретними функціональними можливостями програмного забезпечення. Воно забезпечує відповідність системи очікуванням користувачів та специфікаціям [2].

Модульне тестування, або юніт-тестування, є базовим рівнем тестування, що передбачає перевірку окремих модулів або компонентів програми. Кожен модуль тестується окремо, що дозволяє виявити дефекти на ранніх етапах розробки.

Модульне тестування зазвичай автоматизується за допомогою інструментів, таких як JUnit для Java або NUnit для .NET, що підвищує ефективність та швидкість тестування.

Інтеграційне тестування проводиться після модульного тестування і передбачає перевірку взаємодії між модулями або компонентами системи. Основною метою інтеграційного тестування є виявлення дефектів, які можуть виникати внаслідок інтеграції окремих модулів.

Інтеграційне тестування може проводитися в різних формах, таких як "знизу-вгору" (bottom-up), "зверху-вниз" (top-down) або "великий вибух" (big bang), залежно від підходу до інтеграції модулів.

Системне тестування є завершальним етапом функціонального тестування і передбачає перевірку всієї системи в цілому. Основна мета системного тестування – забезпечити, що всі компоненти працюють разом як єдина система і відповідають визначеним вимогам [3].

Системне тестування зазвичай включає в себе регресійне тестування для перевірки, що нові зміни або виправлення не вплинули на вже існуючу функціональність. Це тестування також може охоплювати нефункціональні аспекти, такі як продуктивність, безпека та зручність використання, але головним чином фокусується на функціональності.

Таким чином, функціональне тестування охоплює всі рівні перевірки програмного забезпечення, починаючи від окремих модулів до інтегрованої системи в цілому. Це дозволяє виявити та усунути дефекти на різних етапах розробки, забезпечуючи високу якість кінцевого продукту [3].

1.2.2 Нефункціональне тестування

Перевіряє аспекти, які не стосуються функціональних вимог, такі як продуктивність (оцінка швидкості та ефективності), безпека (перевірка захищеності) та зручність використання (оцінка зручності інтерфейсу) [4].

Нефункціональне тестування зосереджується на перевірці тих аспектів системи, які не стосуються безпосередньо функціональних вимог, але є критично важливими для загальної якості та користувацького досвіду.

Тестування продуктивності (Performance Testing) оцінює швидкість, ефективність та стабільність системи під різними навантаженнями. Воно допомагає визначити, наскільки швидко система реагує на запити користувачів та як вона поводить себе під час пікових навантажень. Методи тестування продуктивності включають [4]:

- навантажувальне тестування (Load Testing) – перевірка системи під передбачуваним навантаженням, щоб оцінити її продуктивність;
- стрес-тестування (Stress Testing) – перевірка системи під екстремальними умовами, щоб визначити її межі та здатність відновлюватися після збою;
- тестування об'ємів (Volume Testing) - оцінка продуктивності системи при обробці великої кількості даних;
- тестування тривалого навантаження (Soak Testing) - перевірка системи під постійним навантаженням протягом тривалого періоду для виявлення потенційних проблем із продуктивністю.

Тестування безпеки (Security Testing) спрямоване на перевірку захищеності системи від потенційних загроз та вразливостей. Це включає перевірку механізмів аутентифікації, авторизації, шифрування даних та захисту від атак. Методи тестування безпеки включають:

- тестування на проникнення (Penetration Testing) – імітація атак на систему для виявлення вразливостей, які можуть бути використані зловмисниками;
- аудит безпеки (Security Audit) – аналіз системи на відповідність стандартам безпеки та політикам організації;
- тестування управління доступом (Access Control Testing) – перевірка коректності механізмів аутентифікації та авторизації, щоб гарантувати, що доступ до ресурсів має лише уповноважені користувачі;
- сканування вразливостей (Vulnerability Scanning) – використання автоматизованих інструментів для виявлення відомих вразливостей у системі.

Тестування зручності використання (Usability Testing) оцінює інтерфейс користувача на предмет його зручності та інтуїтивності. Воно допомагає зрозуміти, наскільки легко користувачам взаємодіяти з системою та виконувати необхідні завдання. Методи тестування зручності використання включають [4]:

- тестування з користувачами (User Testing) – залучення реальних користувачів до взаємодії з системою та спостереження за їхніми діями, щоб визначити проблемні області;
- евристичний аналіз (Heuristic Evaluation) – оцінка інтерфейсу експертами з юзабіліті на основі встановлених принципів та правил;
- когнітивний аналіз (Cognitive Walkthrough) – аналіз кроків, які користувач має виконати для досягнення певної мети, з метою виявлення потенційних труднощів;
- тестування доступності (Accessibility Testing) – перевірка, чи відповідає інтерфейс вимогам доступності для користувачів з обмеженими можливостями.

Нефункціональне тестування є важливою складовою процесу забезпечення якості програмного забезпечення, оскільки воно допомагає виявити та усунути потенційні проблеми, які можуть вплинути на загальну ефективність та безпеку системи.

1.2.3. Автоматизоване тестування

Передбачає використання інструментів для автоматизації процесу тестування. Основні інструменти:

- Selenium для автоматизації тестування веб-додатків;
- JUnit для модульного тестування на Java;
- TestNG для паралельного виконання тестів на Java.

Автоматизоване тестування передбачає використання спеціальних інструментів для автоматизації процесу тестування. Це дозволяє зменшити час на тестування, підвищити точність і відтворюваність тестів, а також знизити витрати на забезпечення якості [5].

Selenium є одним з найпопулярніших інструментів для автоматизації тестування веб-додатків. Він дозволяє створювати скрипти, які можуть автоматично взаємодіяти з веб-сторінками, виконуючи дії, подібні до дій

реальних користувачів. Selenium використовується для функціонального тестування веб-додатків, перевірки крос-браузерної сумісності та автоматизації регресійних тестів [6].

JUnit є популярним фреймворком для модульного тестування на Java. Він дозволяє розробникам писати і виконувати тести, перевіряючи окремі частини коду (модулі) на наявність помилок. JUnit використовується для модульного тестування, де кожен тестовий випадок перевіряє окремий метод або клас [7].

TestNG є фреймворком для тестування, натхненим JUnit, але з розширеними можливостями. Він підтримує паралельне виконання тестів, конфігураційні анотації, залежності між тестами та інші функції, які роблять його потужним інструментом для тестування. TestNG використовується для функціонального тестування, регресійного тестування, інтеграційного тестування та інших видів тестування, де важлива паралельна обробка та складні конфігурації тестів [8].

Таким чином, автоматизоване тестування є важливою складовою забезпечення якості програмного забезпечення, забезпечуючи швидкість, точність та ефективність тестування.

1.2.4. Ручне тестування

Ручне тестування виконується без використання автоматизації і включає підготовку тестових сценаріїв, їх виконання та документування результатів. Воно є важливою частиною процесу забезпечення якості програмного забезпечення, особливо на ранніх стадіях розробки та при перевірці нових функцій. Ручне тестування дозволяє тестувальникам безпосередньо взаємодіяти з системою, імітуючи дії кінцевих користувачів, що сприяє виявленню дефектів, які можуть бути пропущені автоматизованими тестами. Основні етапи ручного тестування перераховані нижче [9].

Підготовка тестових сценаріїв включає [9]:

- аналіз вимог: ознайомлення з технічними та функціональними вимогами до системи для розуміння, які аспекти потрібно перевіряти;
- розробка тестових сценаріїв: створення докладних описів тестових випадків, включаючи кроки для виконання, очікувані результати та критерії проходження;
- вибір тестових даних: підготовка набору даних, необхідних для виконання тестів.

Виконання тестових сценаріїв включає [9]:

- виконання тестів: тестувальники вручну виконують тестові сценарії, імітуючи дії користувачів у системі;
- спостереження за поведінкою системи: тестувальники спостерігають за результатами виконання тестів, перевіряючи, чи відповідають фактичні результати очікуваним;
- виявлення та реєстрація дефектів: у разі виявлення невідповідностей або помилок тестувальники реєструють дефекти у системі відстеження помилок.

Документування результатів включає [9]:

- запис результатів тестування: документування результатів кожного тестового сценарію, включаючи пройдені та не пройдені тести;
- звітність про дефекти: створення детальних звітів про виявлені дефекти, включаючи опис проблеми, кроки для її відтворення та пріоритет виправлення;
- аналіз результатів: підготовка підсумкових звітів про результати тестування для аналізу командою розробки та зацікавленими сторонами.

Ручне тестування є невід'ємною частиною процесу забезпечення якості програмного забезпечення. Воно дозволяє виявити дефекти, які можуть бути пропущені автоматизованими тестами, і забезпечує гнучкість при перевірці нових функцій. Незважаючи на свої недоліки, ручне

тестування залишається важливим інструментом для забезпечення високої якості програмного забезпечення.

1.2.5 Димне (смоук) тестування

Димне або смоук-тестування – це метод перевірки основної функціональності програми. Проводиться після кожного збирання (build) і перед детальним тестуванням. Основні цілі: перевірка стабільності збірки, виявлення критичних дефектів на ранніх стадіях [10].

Смоук-тестування є важливим етапом, що дозволяє швидко оцінити, чи готова система до подальшого тестування.

Смоук-тестування допомагає швидко оцінити основну працездатність системи, забезпечуючи раннє виявлення критичних дефектів. Це дозволяє зекономити час та ресурси, запобігаючи детальному тестуванню нестабільної збірки.

Процес включає наступні етапи [10]:

- вибір ключових функцій для тестування;
- швидке виконання тестів після кожного збирання;
- оцінка результатів та прийняття рішення про подальше тестування.

1.3 Переваги та недоліки видів тестування

В таблиці 1.1 наведено порівняльну характеристику всіх видів тестування. Враховуючи обмеженість охоплення та поверхневість смоук-тестування, його було обрано через можливість швидкої оцінки основної працездатності системи та виявлення критичних дефектів, що робить його ідеальним для початкових етапів тестування.

Таблиця 1.1 – Преваги та недоліки різних видів тестування

Вид тестування	Певаги	Недоліки
Функціональне тестування	Забезпечує відповідність функціональним вимогам. Допомогає виявити критичні дефекти на ранніх стадіях.	Може бути трудомістким та часозатратним. Вимагає детального розуміння специфікацій.
Нефункц-не тестування	Забезпечує стабільність та безпеку системи. Оцінює користувацький досвід.	Може вимагати спец-них інструментів та знань. Трудомісткість у налаштуванні та проведенні тестів.
Автоматизоване тестування	Зниження витрат часу на тестування. Висока точність та відтворюваність тестів.	Високі початкові витрати на впровадження. Потребує технічних знань та досвіду.
Ручне тестування	Гнучкість у тестуванні нових функцій. Можливість виявлення нефункц-них дефектів, які можуть бути пропущені автоматиз-ми тестами.	Трудомісткість та ризик людських помилок. Тривалий час на проведення тестів.
Димне тестування	Тести виконуються швидко, забезпечуючи оперативну оцінку стабільності збірки. Дозволяє виявити критичні проблеми на ранніх етапах. Ефективність: Запобігає витраті часу на детальне тестування некоректних збірок.	Не покриває всі функції системи. Виявляє лише очевидні дефекти, не даючи детального уявлення про якість продукту.

1.4 Висновки за розділом

Тестування є невід'ємною частиною процесу розробки програмного забезпечення. Різні види тестування мають свої переваги та недоліки, і їх вибір залежить від специфіки проекту та вимог до якості.

Функціональне тестування гарантує правильність виконання заявлених функцій, тоді як нефункціональне забезпечує перевірку інших важливих аспектів, таких як продуктивність і безпека. Автоматизоване тестування дозволяє швидко та ефективно перевіряти великі обсяги коду, знижуючи ризик людських помилок, тоді як ручне тестування є важливим для перевірки нових функцій та користувацького досвіду. Смоук-тестування відіграє важливу роль на початкових етапах тестування, забезпечуючи раннє виявлення критичних дефектів та економію ресурсів.

Оптимальна комбінація різних видів тестування дозволяє досягти високої якості програмного забезпечення, знижуючи ризики і підвищуючи задоволення користувачів. Ефективне тестування є запорукою успішного випуску продукту, який відповідає очікуванням ринку і забезпечує стабільну та безпечну роботу.

Для даної роботи було обрано саме димне тестування через його швидкість оскільки розробки на проектах зазвичай ведуться дуже швидко і нові збірки програм виходять скоріше ніж можна виконати або написати детальний тест. В поєднанні з автоматизованим тестування це дасть швидкий і надійний результат тестування основного критичного функціоналу для кожної збірки.

2 ОГЛЯД ТА ВИБІР ТЕХНОЛОГІЙ ТА ІНСТРУМЕНТІВ РОЗРОБКИ

2.1. Мова програмування

Java і Python є об'єктно-орієнтованою мовами програмування, які широко використовується для розробки різноманітного програмного забезпечення, включаючи автоматизоване тестування.

2.1.1 Переваги Java

Мова Java має ряд переваг для написання автотестів [11]:

- багатоплатформність – Java-код може виконуватися на будь-якій платформі, яка підтримує JVM (Java Virtual Machine), що робить її дуже гнучкою для крос-платформного тестування;
- багатий набір бібліотек – має великий вибір бібліотек і фреймворків для автоматизованого тестування, таких як Selenium, TestNG, JUnit та інші;
- об'єктно-орієнтоване програмування (ООП) – Java підтримує об'єктно-орієнтоване програмування, що дозволяє створювати модульний, зрозумілий та підтримуваний код;
- підтримка спільноти – величезна спільнота розробників та велика кількість ресурсів (документація, форуми, tutorіали), що полегшує вирішення проблем та навчання;
- інтеграція з CI/CD інструментами – Java легко інтегрується з інструментами безперервної інтеграції та доставки (CI/CD), такими як Jenkins, GitLab CI, Travis CI та інші.

2.1.2 Переваги Python

Переваги мови Python [12]:

- простота синтаксису – Python має простий та зрозумілий синтаксис, що робить його легким для вивчення та використання;

- швидкість розробки – завдяки своєму синтаксису Python дозволяє швидко писати та читати код, що прискорює процес розробки тестів;
- великий вибір бібліотек – Python також має широкий набір бібліотек для автоматизованого тестування, таких як PyTest, unittest, та бібліотеки для веб-тестування, такі як Selenium.

2.1.3 Порівняння мов програмування для створення автотестів

Переваги Java над Python для автоматизованого тестування [13]:

- продуктивність – Java зазвичай забезпечує вищу продуктивність виконання у порівнянні з Python, що може бути критично важливим для великих і складних проектів;
- статична типізація – Java є мовою зі статичною типізацією, що дозволяє виявляти помилки на етапі компіляції, зменшуючи кількість помилок під час виконання та підвищуючи надійність коду;
- масштабованість – Java краще підходить для великих проектів з високими вимогами до масштабованості та продуктивності;
- багатоплатформність – хоча Python також підтримує крос-платформенність, Java має більш усталену екосистему для роботи на різних операційних системах та платформах;
- підтримка корпоративного середовища – Java широко використовується в корпоративному середовищі завдяки своїм можливостям для розробки складних, масштабованих і надійних додатків, це робить її кращим вибором для великих компаній та проектів з високими вимогами до надійності та продуктивності.

Таким чином, хоча Python має свої переваги, Java є більш потужною та продуктивною мовою для автоматизованого тестування, особливо для великих проектів з високими вимогами до продуктивності, масштабованості та надійності.

2.2 Вибір допоміжного інтерфейса для написання автотестів

2.2.1 Інструмент Selenium

Selenium є популярним інструментом для автоматизованого тестування веб-додатків. Він дозволяє створювати скрипти для тестування функціональності веб-сайтів на різних браузерах і платформах. Selenium підтримує багато мов програмування, включаючи Java, Python, C#, Ruby та JavaScript, що робить його гнучким і універсальним інструментом. Основні компоненти Selenium включають Selenium WebDriver, Selenium Grid та Selenium IDE [6].

Selenium WebDriver є основним компонентом Selenium, який забезпечує програмний інтерфейс для взаємодії з веб-браузерами. Він дозволяє автоматизувати браузерні дії, такі як кліки, введення тексту, навігація між сторінками тощо [6].

WebDriver безпосередньо взаємодіє з браузерами, що забезпечує високу точність і швидкість виконання тестів. Він підтримує багато браузерів, включаючи Chrome, Firefox, Safari, Edge та інші.

Selenium Grid дозволяє розподіляти виконання тестів на декілька машин, що дозволяє паралельно запускати тести на різних платформах і браузерах.

Grid використовується для прискорення тестування, що особливо важливо для великих тестових наборів. Він забезпечує масштабованість та ефективне використання ресурсів.

Selenium IDE (Integrated Development Environment) є інструментом для запису, редагування і відлагодження тестів. Він працює як плагін для браузерів Firefox і Chrome. IDE дозволяє швидко створювати тестові сценарії без необхідності програмування. Це корисно для створення простих тестів або для швидкої перевірки функціональності [6].

Переваги Selenium [14]:

- багатоплатформність та підтримка різних браузерів – Selenium підтримує тестування на багатьох браузерах і платформах, що робить його універсальним інструментом для крос-браузерного тестування;
- підтримка різних мов програмування – Selenium дозволяє писати тести на різних мовах програмування, що забезпечує гнучкість для команд розробників;
- масштабованість – за допомогою Selenium Grid можна масштабувати тестування, розподіляючи тести на декілька машин і виконуючи їх паралельно;
- інтеграція з іншими інструментами – Selenium легко інтегрується з іншими інструментами та фреймворками, такими як TestNG, JUnit, Maven, Jenkins та інші.

Недоліки Selenium [14]:

- вимагає більше часу на налаштування і конфігурацію;
- тести можуть виконуватися повільніше порівняно з іншими інструментами;
- відсутність вбудованих інструментів для дебагінгу тестів.

2.2.2 Cypress

Для порівняння розглянемо Cypress, який також є популярним інструментом для автоматизованого тестування веб-додатків.

Cypress – це сучасний інструмент для автоматизації тестування веб-додатків, розроблений з акцентом на швидкість та зручність використання. Cypress виконує тести безпосередньо в браузері, що забезпечує високу швидкість і точність [15].

Переваги Cypress [15]:

- простота налаштування – Cypress легко встановлюється і налаштовується, що дозволяє швидко розпочати автоматизоване тестування;

- швидкість виконання тестів – Cypress виконує тести дуже швидко, завдяки своїй архітектурі, яка дозволяє взаємодіяти безпосередньо з браузером;

- зручний дебагінг – Cypress надає потужні інструменти для відлагодження тестів, включаючи реальне відображення виконання тестів у браузері.

Недоліки Cypress [15]:

- підтримує лише мову програмування JavaScript;
- обмежена підтримка браузерів – підтримує лише сучасні браузери, такі як Chrome, Firefox, Edge;
- обмежена масштабованість – відсутність підтримки розподілу виконання тестів на декількох машинах, як у Selenium Grid.

2.2.3 Порівняння Selenium та Cypress

Виходячи з порівняння бачимо, що Selenium підтримує більше браузерів, включаючи менш популярні та старі версії, що забезпечує кращу крос-браузерну сумісність. На відміну від Cypress, який підтримує лише JavaScript, Selenium підтримує кілька мов програмування, таких як Java, Python, C# та інші. Selenium Grid дозволяє легко масштабувати тестування на великі проекти, що може бути важливим для великих команд та складних середовищ. Selenium має більшу спільноту користувачів, що забезпечує більше ресурсів для навчання та вирішення проблем [16].

Вибір між Selenium і Cypress залежить від конкретних потреб і вимог проекту. Selenium є більш універсальним і підтримує більшу кількість мов програмування та браузерів, тоді як Cypress забезпечує швидкість, простоту налаштування та зручний дебагінг, але обмежується лише JavaScript і підтримкою сучасних браузерів.

В таблиці 2.1 наведено порівняння цих двох інструментів.

Таблиця 2.1 – Порівняння Selenium та Cypress

Характеристика	Selenium	Cypress
Мови програмування	Java, Python, C#, Ruby, JavaScript та інші	JavaScript
Підтримка браузерів	Chrome, Firefox, Safari, Edge, Opera	Chrome, Firefox, Edge
Швидкість виконання тестів	Помірна	Висока
Налаштування та конфігурація	Більш складне, вимагає більше часу	Легке та швидке налаштування
Дебагінг	Менш зручний, відсутність вбудованих інструментів	Зручний, вбудовані інструменти для дебагінгу
Автоматичне очікування	Необхідно вручну писати код для очікування	Автоматичне очікування на завантаження елементів
Масштабованість	Підтримка Selenium Grid для розподіленого виконання	Відсутність підтримки розподіленого виконання

Таким чином, Selenium залишається одним із найпотужніших та найгнучкіших інструментів для автоматизованого тестування веб-додатків, що робить його кращим вибором для багатьох проектів, особливо тих, які вимагають широкої підтримки браузерів і платформ.

2.3. Вибір середовища розробки

2.3.1 Середовище розробки IntelliJ IDEA

IntelliJ IDEA є інтегрованим середовищем розробки (IDE), що підтримує мову програмування Java та багато інших мов. Це середовище розробки забезпечує зручний інтерфейс, потужні інструменти для розробки,

налагодження та тестування коду, а також інтеграцію з системами контролю версій та іншими інструментами [17].

Основні переваги IntelliJ IDEA перераховані нижче.

Підтримка різних мов програмування. Крім Java, IntelliJ IDEA підтримує мови, такі як Kotlin, Groovy, Scala, Python, JavaScript, TypeScript та багато інших. Це робить його універсальним інструментом для багатомовних проектів.

Розширені інструменти для розробки. IntelliJ IDEA надає розширені можливості для рефакторингу, аналізу коду, генерації коду та автоматичного завершення коду. Ці інструменти підвищують продуктивність розробників і допомагають зберігати якість коду.

Налагодження та тестування. IDE включає потужний налагоджувач, який підтримує багатопоточне налагодження, гарячі заміни коду (hot swaps) та віддалене налагодження. Інструменти для тестування включають підтримку JUnit, TestNG та інших тестових фреймворків [17].

Інтеграція з системами контролю версій. IntelliJ IDEA підтримує інтеграцію з популярними системами контролю версій, такими як Git, SVN, Mercurial та Perforce. Це дозволяє легко керувати версіями коду, робити коміти, створювати гілки та зливати зміни.

Підтримка CI/CD інструментів. IDE інтегрується з інструментами безперервної інтеграції та доставки, такими як Jenkins, TeamCity та Bamboo, що дозволяє автоматизувати процеси розгортання та тестування.

Плагіни та розширення. IntelliJ IDEA має багату екосистему плагінів, які дозволяють розширювати функціональність IDE відповідно до потреб проекту. Це включає підтримку різних фреймворків, баз даних, інструментів для розробки фронтенду та багато іншого.

Інтелектуальний інтерфейс користувача. IntelliJ IDEA надає інтелектуальні підказки, які допомагають розробникам швидше писати код та виявляти помилки на ранніх етапах [17].

2.3.2 Порівняння IntelliJ IDEA з середовищем розробки Eclipse

Для порівняння розглянемо Eclipse, яке також є популярним інтегрованим середовищем розробки для Java.

Переваги Eclipse [17]:

- безкоштовність – Eclipse є повністю безкоштовним інструментом з відкритим кодом;
- велика кількість плагінів – Eclipse має широку підтримку плагінів, які дозволяють налаштовувати IDE для різних типів проектів;
- підтримка багатьох мов програмування, як і IntelliJ IDEA, Eclipse підтримує різні мови програмування через плагіни.

IntelliJ IDEA та Eclipse є потужними інструментами для розробки автотестів, але кожен з них має свої переваги та недоліки.

IntelliJ IDEA надає більш зручний та інтуїтивний інтерфейс, потужне автодоповнення коду, легку інтеграцію з сучасними фреймворками та інструментами для CI/CD. Він також має кращу підтримку інструментів для налагодження та рефакторингу коду. Комерційна версія (Ultimate) надає додаткові можливості для професійної розробки.

Eclipse є безкоштовним та з відкритим кодом, що робить його доступним для всіх. Він має широку підтримку плагінів, що дозволяє розширювати функціональність, але може вимагати більше часу на налаштування та конфігурацію. Eclipse також має хорошу підтримку різних мов програмування через плагіни [17].

Наглядне порівняння наведено в таблиці 2.2.

Вибір між IntelliJ IDEA та Eclipse залежить від ваших потреб і уподобань. Якщо вам потрібен інструмент з потужними можливостями та зручним інтерфейсом, IntelliJ IDEA може бути кращим вибором. Якщо ж ви шукаєте безкоштовний інструмент з великою кількістю розширень і не проти витратити більше часу на налаштування, Eclipse також є відмінним варіантом.

Таблиця 2.2 – Порівняння середовищ розробки

Характеристика	IntelliJ IDEA	Eclipse
Підтримка мов програмування	Широка підтримка мов (Java, Kotlin, Groovy, Scala, Python, JavaScript, TypeScript та інші)	Підтримка багатьох мов через плагіни (Java, C/C++, Python, PHP та інші)
Інтерфейс користувача	Інтуїтивно зрозумілий, зручний та сучасний інтерфейс	Інтерфейс може бути складнішим для новачків, менш інтуїтивний
Інструменти для розробки тестів	Підтримка JUnit, TestNG, Selenium, Cucumber з коробки	Підтримка JUnit, TestNG, Selenium, Cucumber через плагіни
Налаштування та конфігурація	Легке налаштування та конфігурація, інтуїтивно зрозумілі меню	Потребує більше налаштувань, інтерфейс менш зручний для конфігурації
Автодоповнення коду	Потужне автодоповнення коду, підтримка рефакторингу	Хороше автодоповнення, але може поступатися IntelliJ IDEA
Швидкість та продуктивність	Висока продуктивність, швидкий запуск та виконання	Може бути повільнішим, особливо з великою кількістю плагінів
Підтримка фреймворків	Широка підтримка сучасних фреймворків (Spring, Hibernate, Maven, Gradle)	Широка підтримка через плагіни, але може вимагати більше налаштувань

Таким чином, IntelliJ IDEA є потужним та зручним інструментом для розробки на Java, який забезпечує високу продуктивність та ефективність

роботи розробників завдяки своїм розширеним можливостям та інтеграції з сучасними інструментами та технологіями.

2.4 Висновки за розділом

У цьому розділі було здійснено огляд та вибір технологій і інструментів для розробки автотестів. Вивчено переваги та недоліки мов програмування Java і Python, а також порівняно їх з точки зору застосування для автоматизованого тестування.

Java було вибрано як основну мову для написання автотестів завдяки її багатоплатформності, багатому набору бібліотек, об'єктно-орієнтованому підходу та високій продуктивності.

Для написання автотестів обрано Selenium, що забезпечує гнучкість, масштабованість і підтримку різних мов програмування та браузерів. Описано переваги та недоліки Selenium, а також порівняно його з Cypress, що також є популярним інструментом для автоматизованого тестування веб-додатків.

IntelliJ IDEA було обрано як середовище розробки завдяки його потужним інструментам, підтримці різних мов програмування, інтеграції з системами контролю версій та інструментами CI/CD. Порівняно IntelliJ IDEA з Eclipse, відзначено переваги та недоліки обох середовищ розробки.

Таким чином, обрані технології та інструменти забезпечують ефективну розробку автотестів, що відповідають вимогам проекту, та дозволяють здійснити якісне димне тестування сайту.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДИМНОГО ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ

Для розробки програмного забезпечення димного тестування обирається підхід, що передбачає використання Java та Selenium. Це дозволяє забезпечити високу продуктивність та гнучкість тестування, а також інтеграцію з існуючими системами безперервної інтеграції. Використання Selenium WebDriver забезпечить доступ до різних браузерів, що необхідно для перевірки кросбраузерної сумісності.

Димне тестування буде виконуватись для сайту <https://go.givingli.com>. Сайт Givingli є платформою для надсилання цифрових подарунків та листівок. Основними функціями сайту є створення та персоналізація листівок (рис. 3.1), вибір подарунків (рис.3.2), управління акаунтом користувача та здійснення платежів.

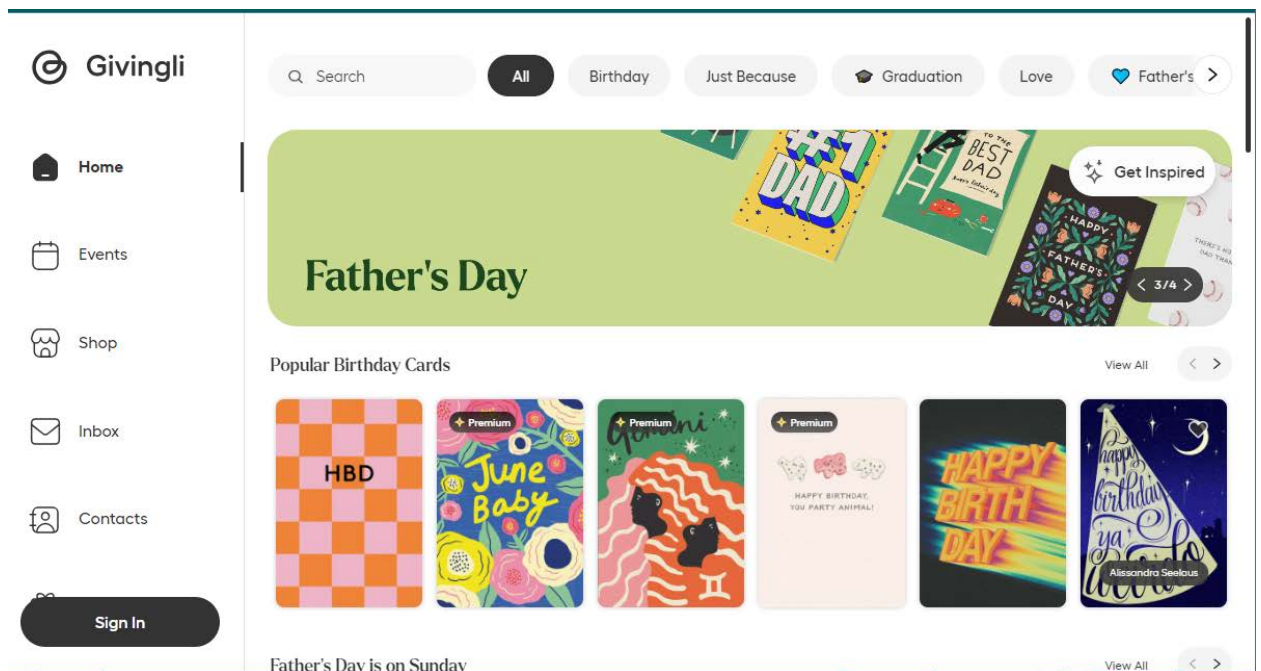


Рисунок 3.1 – Головна сторінка сайту вибору листівок

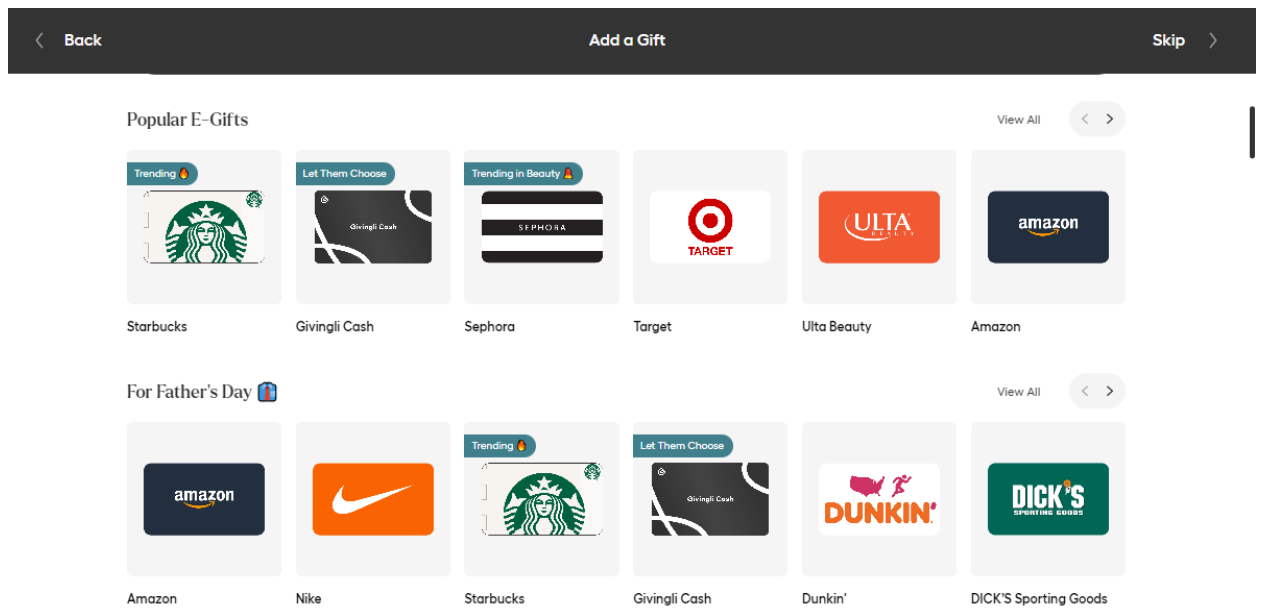


Рисунок 3.2 – Вибір подарункових карток

3.1. Визначення тестових випадків

Для проведення смоук-тестування будуть розроблені наступні тестові випадки:

- реєстрація нового користувача, перевірка можливості створення нового облікового запису;
- вхід користувача, перевірка можливості входу до системи з використанням існуючого облікового запису;
- створення листівки, перевірка процесу створення та персоналізації листівки;
- відправка листівки, перевірка можливості відправки створеної листівки;
- вибір та купівля подарунка, перевірка функціональності вибору та покупки подарунка;
- здійснення платежу, перевірка процесу здійснення оплати за послуги.

3.2. Реалізація тестових випадків

Для кожного тестового випадку буде розроблено окремий Java-клас, що реалізує відповідний тест. В додатку А наведено приклад коду для тесту реєстрації нового користувача.

Для покращення підтримки та читабельності коду буде використано паттерн Page Object Model. Це дозволить створити окремі Java-класи для кожної сторінки сайту, що інкапсують логіку взаємодії з елементами сторінки.

3.3 Структура автотестів

Для розробки автотестів використовується Java та бібліотека Selenium. Структура проекту в IntelliJ IDEA включає папки та файли зображені на рисунку 3.3.

У структурі проекту, файли у папках `src/main/java` та `test/java` мають різні призначення.

Папка `src/main/java` містить основний код додатку. Тут розміщуються файли, які реалізують бізнес-логіку, моделі даних, сервіси, контролери та інші компоненти, які складають вебзастосунок.

Папка `test/java` містить тестовий код. Тут розміщуються файли, які реалізують тестові випадки для перевірки коректності роботи основного коду. Тестовий код використовується для автоматизованого тестування функціональності додатку.

Обидві папки допомагають організувати код проекту таким чином, щоб забезпечити розділення основного функціоналу та тестів, що робить проект більш структурованим і легким для підтримки.

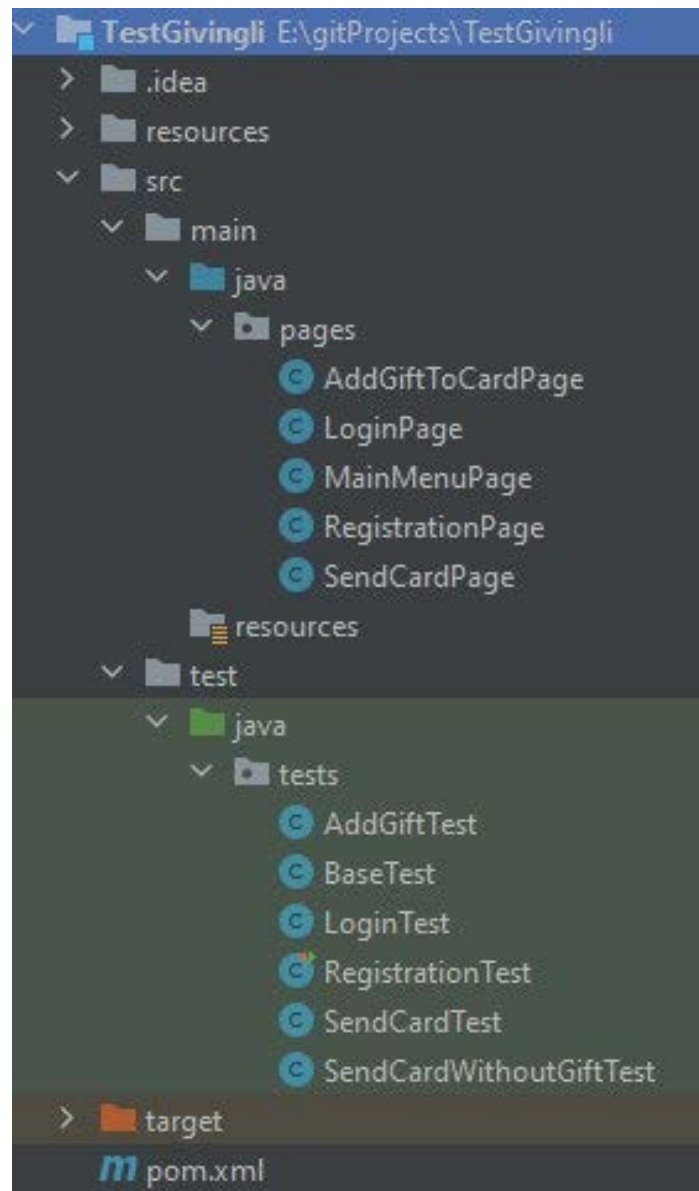


Рисунок 3.3 – Структура проекту автотестів

3.4 Опис автотестів

В таблиці 3.1 наведено короткий опис розроблених автотестів для димного тестування вебзастосунку.

LoginTest.java – тест виконує вхід користувача на сайт, перевіряючи коректність введення логіну та паролю, та підтверджує, що користувач успішно увійшов до системи.

RegisterTest.java – тест виконує процес реєстрації нового користувача, заповнюючи всі необхідні поля, та перевіряє успішне створення облікового запису.

AddGiftTest.java – тест перевіряє процес додавання е-гіфта до створеної картки, включаючи вибір е-гіфта та його успішне додавання до картки.

SendCardTest.java – тест перевіряє повний процес відправки картки з доданим е-гіфтом, включаючи заповнення форми одержувача та підтвердження успішної відправки картки.

SendCardWithoutGiftTest.java – тест перевіряє процес відправки картки без доданого е-гіфта, включаючи заповнення необхідних даних та підтвердження успішного відправлення картки.

Таблиця 3.1 – Опис автотестів

Назва	Опис
LoginTest.java:	Перевірка успішного входу до системи
RegisterTest.java:	Перевірка успішної реєстрації нового користувача.
AddGiftTest.java	Перевірка додавання е-гіфта до картки.
SendCardTest.java	Перевірка відправки картки з е-гіфтом
SendCardWithoutGiftTest.java	Перевірка відправки картки без е-гіфта.

3.5 Алгоритми роботи автотестів

На рисунках 3.4 – 3.8 зображені алгоритми роботи автотестів у вигляді діаграм послідовності дій.

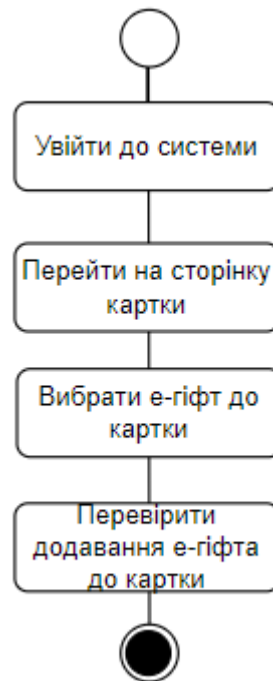


Рисунок 3.4 – Діаграма послідовності виконання тесту AddGiftTest.java

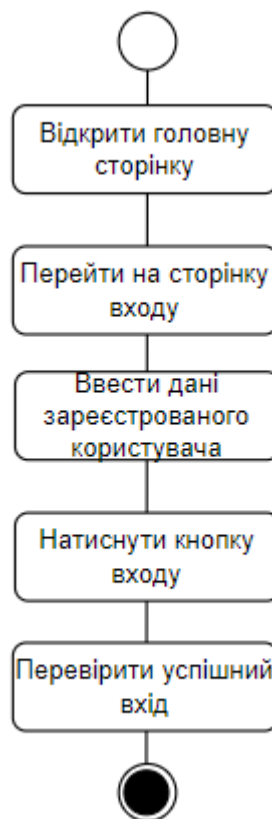


Рисунок 3.5 – Діаграма послідовності виконання тесту LoginTest.java

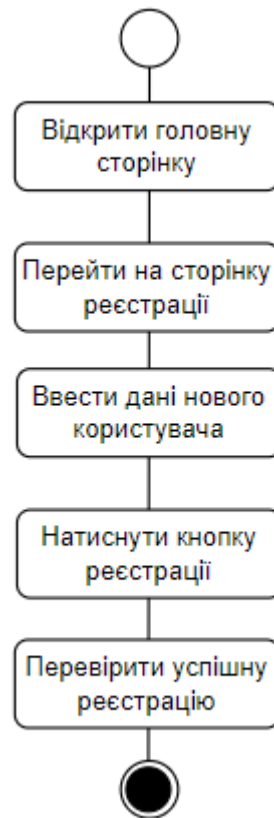


Рисунок 3.6 – Діаграма послідовності виконання тесту RegisterTest.java

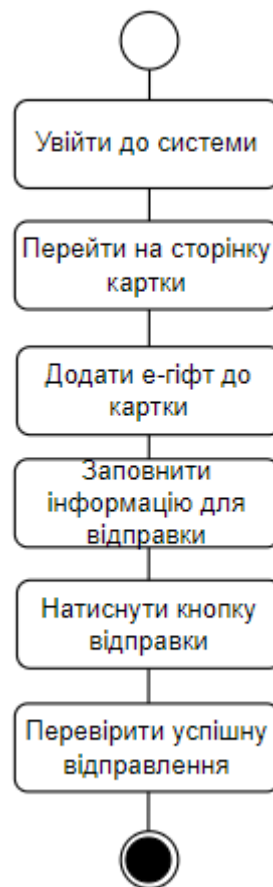


Рисунок 3.7 – Діаграма послідовності виконання тесту SendCardTest.java

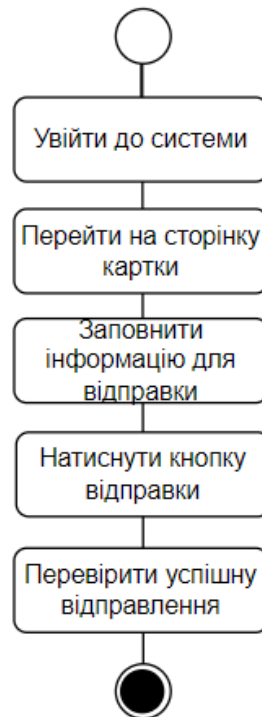


Рисунок 3.8 – Діаграма послідовності виконання тесту
SendCardWithoutGiftTest.java

При логіні та вході в системи крок де необхідно ввести дані зареєстрованого користувача, мається на увазі номер телефону та код.

Для реєстрації нового користувача необхідно ввести такі дані нового користувача: номер телефону, код, ім'я, прізвище, пошта, та дата народження.

Для листівок з е-гіфтом (е-гіфт це картка на якусь сумму на покупку в конкретному магазині або сервісі і.т.п.) необхідно заповнити інформацію для відправки, а саме: назва листівки, пошта або телефон отримувача, данні банківської картки для покупки е-гіфта.

Майже теж саме необхідно заповнити для відправки просто листівки, просто не треба додавати номер картки оскільки відправлення листівки безкоштовне.

3.6 Висновки до розділу

У цьому розділі було описано процес розробки програмного забезпечення для димного тестування вебзастосунку на прикладі сайту Givingli. Було обрано інструменти Java та Selenium для реалізації тестування, що забезпечує високу продуктивність та гнучкість. Розглянуто основні функції сайту та визначено тестові випадки для перевірки критичних шляхів використання.

Були розроблені та описані наступні автотести: LoginTest.java, RegisterTest.java, AddGiftTest.java, SendCardTest.java, SendCardWithoutGiftTest.java.

Також було розглянуто структуру проекту, яка розділяє основний код додатку та тестовий код, що робить проект більш структурованим та легким для підтримки. Описані алгоритми роботи кожного автотесту, що забезпечує чітке розуміння послідовності дій при виконанні тестів.

Розроблені автотести охоплюють основні функції сайту, що дозволяє швидко та ефективно проводити димне тестування критичної функціональності перед кожною заливкою нового функціонала.

4 ІНСТРУКЦІЇ ПО ЗАПУСКУ АВТОТЕСТІВ

Інструкцію по налаштуванню та запуску автотестів можна представити у вигляді діаграми послідовності цій, яка зображена на рисунку 4.1. Всі кроки детально описані далі.

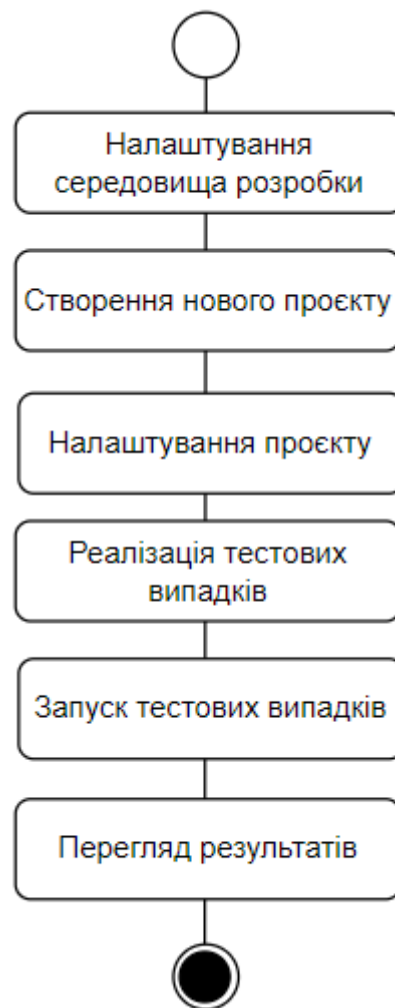


Рисунок 4.1 – Діаграма послідовності налаштування та запуску автотестів

4.1 Налаштування середовища розробки

Початкове налаштування середовища розробки включає в себе наступні кроки описані нижче в цьому розділі.

Встановлений Java Development Kit (JDK). Переконайтеся, що JDK встановлено та налаштовано в системі. Перевірте встановлену версію JDK набравши команду «java -version» у панелі команд.

Встановлений IntelliJ IDEA. Для розробки автотестів використовується IntelliJ IDEA. Основні кроки включають встановлення IntelliJ IDEA, налаштування проекту та додавання необхідних бібліотек. Переконайтеся, що у вас встановлена IDE IntelliJ IDEA.

Встановлений веб-драйвер (ChromeDriver, GeckoDriver тощо). Завантажте відповідний драйвер для браузера, який ви будете використовувати для тестування. Завантажте ChromeDriver з [офіційного сайту](<https://sites.google.com/a/chromium.org/chromedriver/downloads>) (для тестування у браузері Chrome).

Створений обліковий запис на GitHub або іншому репозиторії коду. Для збереження та управління кодом.

4.2 Створення нового проєкту в IntelliJ IDEA

Відкрийте IntelliJ IDEA та створіть новий проєкт:

- виберіть "Create New Project";
- оберіть "Java" та переконайтеся, що вказаний коректний JDK.

Додайте залежності для Selenium та TestNG:

- створіть файл `pom.xml` для управління залежностями за допомогою Maven;
- синхронізуйте проєкт з Maven.

4.3 Налаштування проєкту

Додайте веб-драйвер до системного шляху. Додайте шлях до ChromeDriver в змінні середовища (PATH).


```

BUILD SUCCESSFUL in 23s
3 actionable tasks: 3 executed

Build Analyzer results available
14:41:17: Execution finished ':test --tests "tests.MainMenuTests.testGoToAnotherMenuIconNotSelected"'.

```

Рисунок 4.3 – Результат виконання тесту

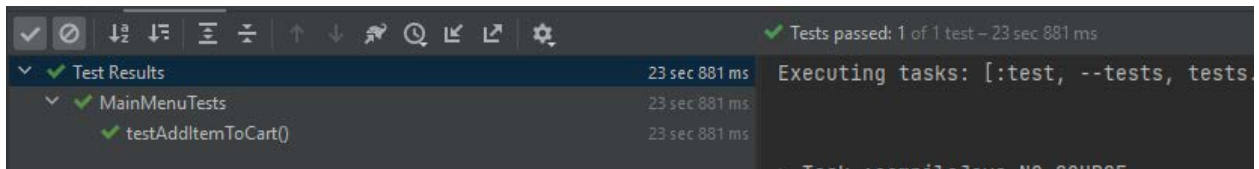


Рисунок 4.4 - Процес виконання тесту перевірки додавання е-гіфта до листівки (картки)

```

BUILD SUCCESSFUL in 26s
3 actionable tasks: 1 executed, 2 up-to-date

Build Analyzer results available
15:07:03: Execution finished ':test --tests "tests.MainMenuTests.testAddItemToCart"'.

```

Рисунок 4.5 – Результат тесту

Також можна налаштувати Jenkins для автоматичного запуску тестів для цього необхідно виконати кроки описані нижче.

Завантажте та встановіть Jenkins з [офіційного сайту](<https://www.jenkins.io/download/>).

Запустіть Jenkins і налаштуйте початкові параметри.

Далі необхідно створити нове завдання (Job): у Jenkins створіть нове завдання типу "Freestyle project"; вкажіть Git-репозиторій з вашим проектом.

Наступне є налаштування збірки: у розділі "Build" додайте крок збірки типу "Invoke top-level Maven targets"; вкажіть ціль збірки `clean test`.

І останнє це запуск тестів: збережіть завдання та запустіть його вручну для перевірки; налаштуйте автоматичний запуск після кожного коміту в репозиторій.

Тепер все готово до автоматизованого димного тестування сайту [Givingli](<https://go.givingli.com>) з використанням Java, Selenium і IntelliJ IDEA.

4.5 Висновки за розділом

У цьому розділі було детально описано процес налаштування та запуску автотестів для димного тестування вебзастосунку. Було розглянуто налаштування середовища розробки, включаючи встановлення необхідних інструментів та залежностей.

Описано процес створення нового проєкту в IntelliJ IDEA та налаштування його структури відповідно до паттерну Page Object Model, що забезпечує підтримуваність і масштабованість коду.

Розглянуто реалізацію тестових випадків та запуск автотестів у середовищі розробки, а також налаштування Jenkins для автоматичного запуску тестів.

Ці кроки забезпечують ефективне димне тестування, що дозволяє виявляти критичні помилки на ранніх етапах розробки та підвищувати якість програмного забезпечення.

ВИСНОВКИ

В результаті дипломної роботи було розроблено програму для автоматизованого димного тестування вебзастосунку сайту Givingli.

Java була обрана завдяки своїм перевагам у продуктивності, багатоплатформності, великому набору бібліотек та підтримці об'єктно-орієнтованого програмування. Selenium WebDriver забезпечує гнучкість і масштабованість, підтримку різних браузерів і інтеграцію з інструментами безперервної інтеграції (CI/CD). IntelliJ IDEA була обрана за зручний інтерфейс, потужні інструменти для розробки та підтримку різних мов програмування.

Розроблено структуру проєкту та алгоритми виконання тестових випадків та розроблено автотести на мові Java для основних функцій сайту: реєстрація, вхід, створення та відправка листівок, додавання е-гіфта.

Описано налаштування середовища розробки, створення нового проєкту в IntelliJ IDEA та налаштування структури директорій.

Розроблено детальні інструкції по реалізації та запуску автотестів.

Димне тестування дозволило швидко оцінити основну працездатність системи та виявити критичні дефекти на ранніх етапах розробки.

Автоматизовані тести забезпечили високу швидкість і ефективність перевірки, знижуючи ризик людських помилок.

Таким чином, були виконання всі поставленні завдання та розроблене програмне забезпечення автоматизованого димного тестування суттєво підвищило якість і надійність вебзастосунку, що дозволяє швидко реагувати на зміни та забезпечувати стабільну роботу сайту Givingli.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. What is testing [Electronic resource]. – Access mode: chrome-extension://efaidnbmnnnibpcajpcgclefindmkaj/https://tmcpfs.ops.fhwa.dot.gov/cfp/projects/uploaded_files/Final%20Q&A.pdf.
2. Functional testing [Electronic resource]. – Access mode: <https://www.geeksforgeeks.org/software-testing-functional-testing>.
3. What is functional testing [Electronic resource]. – Access mode: <https://www.opentext.com/what-is/functional-testing#:~:text=Related%20products-,Overview,with%20the%20end%20user's%20expectations>.
4. Non functional testing [Electronic resource]. – Access mode: <https://www.guru99.com/non-functional-testing.html>.
5. Automation testing [Electronic resource]. – Access mode: <https://www.geeksforgeeks.org/automation-testing-software-testing/?ref=lbp>.
6. Selenium [Electronic resource]. – Access mode: <https://www.selenium.dev>.
7. JUnit documentation [Electronic resource]. – Access mode: [Electronic resource]. – Access mode: <https://junit.org/junit5>.
8. TestNG <https://testng.org>.
9. Manual testing [Electronic resource]. – Access mode: <https://www.geeksforgeeks.org/software-testing-manual-testing/?ref=lbp>.
10. Smoke testing [Electronic resource]. – Access mode: <https://www.geeksforgeeks.org/smoke-testing-software-testing>.
11. Test automation with Java [Electronic resource]. – Access mode: <https://www.qamadness.com/knowledge-base/test-automation-with-java>.
12. Python automation benefits [Electronic resource]. – Access mode: <https://www.linkedin.com/pulse/python-automation-benefits-koysur-miah>.
13. Java vs Python: which one is better with selenium [Electronic resource]. – Access mode: <https://www.testrigtechnologies.com/blogs/java-vs-python-which-one-is-better-with->

ДОДАТОК А
Текст програми

A.1 BaseTest.java

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeMethod;
import java.util.concurrent.TimeUnit;

public class BaseTest {
    WebDriver driver;

    @BeforeMethod

    public void setUp() {
        System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");
        driver = new ChromeDriver();
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
        driver.manage().window().maximize();
    }

    @AfterMethod

    public void tearDown() {
        driver.quit();
    }
}
```

A.2 LoginTest.java

```
import org.openqa.selenium.By;
```

```
import org.testng.Assert;
import org.testng.annotations.Test;

public class LoginTest extends BaseTest {

    @Test

    public void testLogin() {
        driver.get("https://go.givingli.com/");
        driver.findElement(By.id("login")).click();
        driver.findElement(By.id("email")).sendKeys("user@example.com");
        driver.findElement(By.id("password")).sendKeys("password");
        driver.findElement(By.id("login-button")).click();
        Assert.assertTrue(driver.findElement(By.id("welcome-
message")).isDisplayed());
    }
}
```

A.3 RegisterTest.java

```
import org.openqa.selenium.By;
import org.testng.Assert;
import org.testng.annotations.Test;

public class RegisterTest extends BaseTest {

    @Test

    public void testRegister() {
        driver.get("https://go.givingli.com/");
```

```

driver.findElement(By.id("register")).click();
driver.findElement(By.id("email")).sendKeys("newuser@example.com");
driver.findElement(By.id("password")).sendKeys("password");
driver.findElement(By.id("register-button")).click();
Assert.assertTrue(driver.findElement(By.id("welcome-
message")).isDisplayed());
    }
}

```

A.4 AddGiftTest.java

```

import org.openqa.selenium.By;
import org.testng.Assert;
import org.testng.annotations.Test;

public class AddGiftTest extends BaseTest {

    @Test

    public void testAddGift() {
        driver.get("https://go.givingli.com/");
        driver.findElement(By.id("login")).click();
        driver.findElement(By.id("email")).sendKeys("user@example.com");
        driver.findElement(By.id("password")).sendKeys("password");
        driver.findElement(By.id("login-button")).click();
        driver.findElement(By.id("create-card")).click();
        driver.findElement(By.id("add-gift")).click();
        driver.findElement(By.id("select-gift")).click();
    }
}

```

```

        Assert.assertTrue(driver.findElement(By.id("gift-added-
message")).isDisplayed());
    }
}

```

A.5 SendCardTest.java

```

import org.openqa.selenium.By;
import org.testng.Assert;
import org.testng.annotations.Test;

public class SendCardTest extends BaseTest {

    @Test

    public void testSendCard() {
        driver.get("https://go.givingli.com/");
        driver.findElement(By.id("login")).click();
        driver.findElement(By.id("email")).sendKeys("user@example.com");
        driver.findElement(By.id("password")).sendKeys("password");
        driver.findElement(By.id("login-button")).click();
        driver.findElement(By.id("create-card")).click();
        driver.findElement(By.id("add-gift")).click();
        driver.findElement(By.id("select-gift")).click();
        driver.findElement(By.id("recipient-
email")).sendKeys("recipient@example.com");
        driver.findElement(By.id("send-button")).click();
        Assert.assertTrue(driver.findElement(By.id("sent-message")).isDisplayed());
    }
}

```

A. 6 SendCardWithoutGiftTest.java

```
import org.openqa.selenium.By;
import org.testng.Assert;
import org.testng.annotations.Test;

public class SendCardWithoutGiftTest extends BaseTest {
    @Test
    public void testSendCardWithoutGift() {
        driver.get("https://go.givingli.com/");
        driver.findElement(By.id("login")).click();
        driver.findElement(By.id("email")).sendKeys("user@example.com");
        driver.findElement(By.id("password")).sendKeys("password");
        driver.findElement(By.id("login-button")).click();
        driver.findElement(By.id("create-card")).click();
        driver.findElement(By.id("recipient-
email")).sendKeys("recipient@example.com");
        driver.findElement(By.id("send-button")).click();
        Assert.assertTrue(driver.findElement(By.id("sent-message")).isDisplayed());
    }
}
```

A.7 Створення Page Object для реєстрації

```
package com.example.pages;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;

public class RegistrationPage {
    private WebDriver driver;
```

```
private By signUpButton = By.id("signUpButtonId");
private By usernameField = By.id("usernameFieldId");
private By passwordField = By.id("passwordFieldId");
private By registerButton = By.id("registerButtonId");

public RegistrationPage(WebDriver driver) {
    this.driver = driver;
}

public void open() {
    driver.get("https://go.givingli.com");
}

public void clickSignUp() {
    driver.findElement(signUpButton).click();
}

public void enterUsername(String username) {
    driver.findElement(usernameField).sendKeys(username);
}

public void enterPassword(String password) {
    driver.findElement(passwordField).sendKeys(password);
}

public void clickRegister() {
    driver.findElement(registerButton).click();
}
}
```

А.8 Файл pom

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>selenium-smoke-test</artifactId>
  <version>1.0-SNAPSHOT</version>
  <dependencies>
    <dependency>
      <groupId>org.seleniumhq.selenium</groupId>
      <artifactId>selenium-java</artifactId>
      <version>4.0.0</version>
    </dependency>
    <dependency>
      <groupId>org.testng</groupId>
      <artifactId>testng</artifactId>
      <version>7.4.0</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

ДОДАТОК Б
Слайди презентації

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
Кафедра програмних засобів

Дипломна кваліфікаційна робота на тему:
**Розробка програмного забезпечення для
проведення димного тестування вебзастосунку.
Software Development for Conducting a Smoke
Test of a Web Application**

Виконав ст.гр. КНТ-130

Гарегін АКОПЯН

Керівник: к.т.н., доцент

Ольга ГЛАДКОВА

1

Рисунок Б.1 – Слайд №1

Види тестування

- Функціональне;
- Нефункціональне;
- **Димне;**
- Автоматизоване;
- Ручне.

2

Рисунок Б.2 – Слайд №2

Мета та завдання роботи

Об'єкт дослідження – програмні засоби для автоматизації тестування вебзастосунків.
 Предмет дослідження – програмне забезпечення для автоматизованого димного тестування вебзастосунку.
 Мета роботи – розробка програмного забезпечення для автоматизації димного тестування сайту, що дозволить швидко та ефективно перевіряти основну працездатність сайту після кожного збирання коду.

Для досягнення поставленої мети були вирішені наступні завдання:

- аналіз існуючих видів тестування;
- аналіз теоретичних аспектів смоук тестування;
- розробка архітектури автоматизованої системи димного тестування;
- розробка тестових сценаріїв;
- реалізація автотестів.



Рисунок Б.3 – Слайд №3

Вибір інструментарію розробки

Характеристика	Selenium	Cypress
Мови програмування	Java, Python, C#, Ruby, JavaScript та інші	JavaScript
Підтримка браузерів	Chrome, Firefox, Safari, Edge, Opera	Chrome, Firefox, Edge
Швидкість виконання тестів	Помірна	Висока
Налаштування та конфігурація	Більш складне, вимагає більше часу	Легке та швидке налаштування
Дебагінг	Менш зручний, відсутність вбудованих інструментів	Зручний, вбудовані інструменти для дебагінгу
Автоматичне очікування	Необхідно вручну писати код для очікування	Автоматичне очікування на завантаження елементів
Масштабованість	Підтримка Selenium Grid для розподіленого виконання	Відсутність підтримки розподіленого виконання

Характеристика	IntelliJ IDEA	Eclipse
Підтримка мов програмування	Широка підтримка мов (Java, Kotlin, Groovy, Scala, Python, JavaScript, TypeScript та інші)	Підтримка багатьох мов через плагіни (Java, C/C++, Python, PHP та інші)
Інтерфейс користувача	Інтуїтивно зрозумілий, зручний та сучасний інтерфейс	Інтерфейс може бути складнішим для новачків, менш інтуїтивний
Інструменти для розробки тестів	Підтримка JUnit, TestNG, Selenium, Cucumber з коробки	Підтримка JUnit, TestNG, Selenium, Cucumber через плагіни
Налаштування та конфігурація	Легке налаштування та конфігурація, інтуїтивно зрозумілі меню	Потребує більше налаштувань, інтерфейс менш зручний для конфігурації
Автодоповнення коду	Потужне автодоповнення коду, підтримка рефакторингу	Хороше автодоповнення, але може поступатися IntelliJ IDEA
Швидкість та продуктивність	Висока продуктивність, швидкий запуск та виконання	Може бути повільнішим, особливо з великою кількістю плагінів
Підтримка фреймворків	Широка підтримка сучасних фреймворків (Spring, Hibernate, Maven, Gradle)	Широка підтримка через плагіни, але може вимагати більше налаштувань



Рисунок Б.4 – Слайд №4

Об’єкт тестування

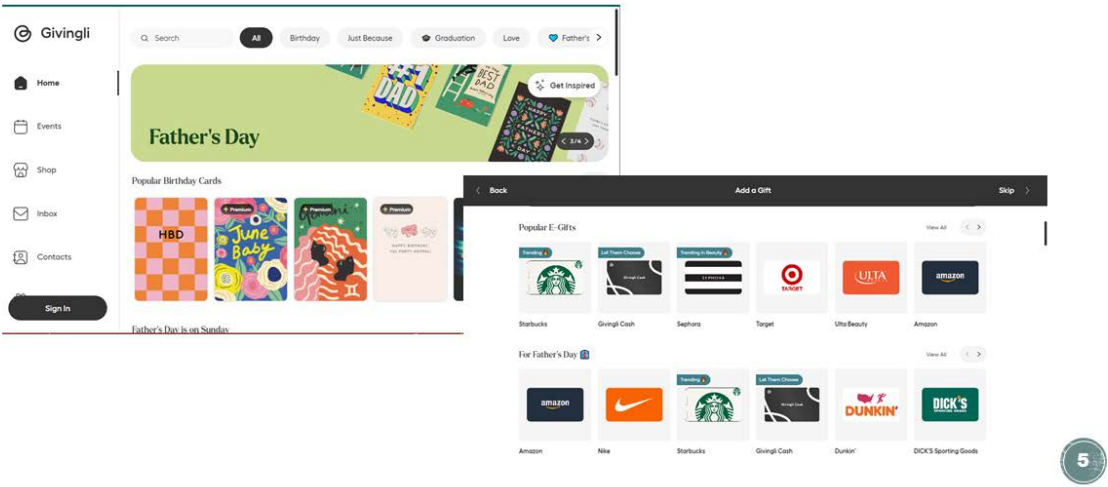
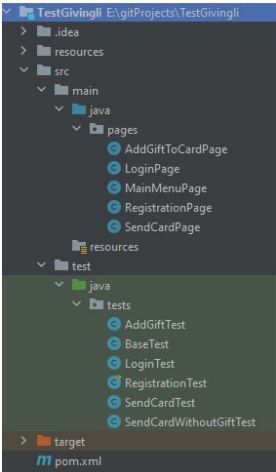


Рисунок Б.5 – Слайд №5

Структура проєкту



Назва	Опис
LoginTest.java:	Перевірка успішного входу до системи
RegisterTest.java:	Перевірка успішної реєстрації нового користувача.
AddGiftTest.java	Перевірка додавання е-гіфта до картки.
SendCardTest.java	Перевірка відправки картки з е-гіфтом
SendCardWithoutGiftTest.java	Перевірка відправки картки без е-гіфта.

Рисунок Б.6 – Слайд №6

Діаграми послідовності виконання автотестів

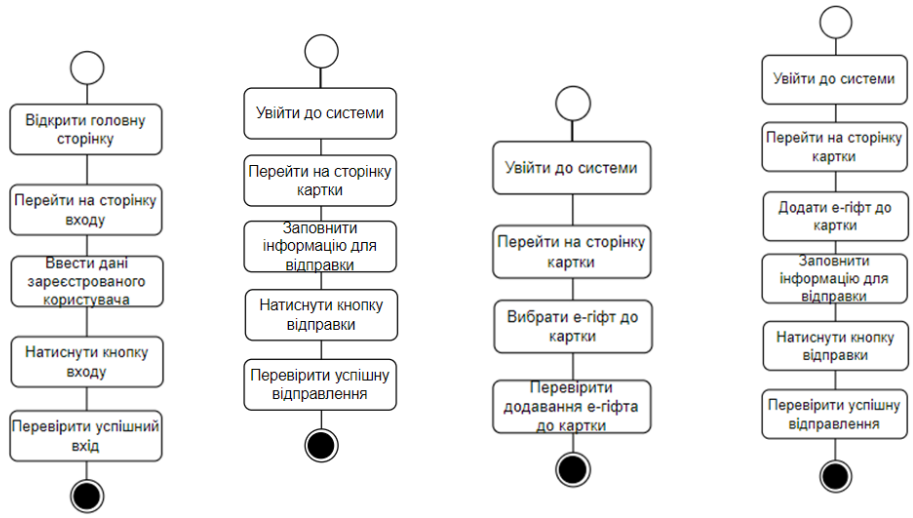


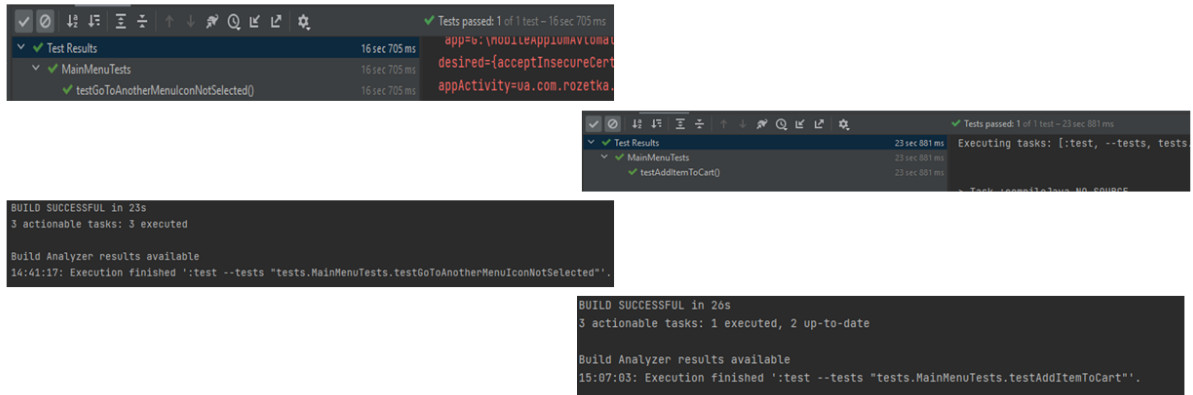
Рисунок Б.7 – Слайд №7

Діаграма послідовності налаштування та запуску автотестів



Рисунок Б.8 – Слайд №8

Результати тестування



9

Рисунок Б.9 – Слайд №9

Висновки

В результаті виконання дипломної роботи було:

- Розроблено структуру проєкту та алгоритми виконання тестових випадків та розроблено автотести на мові Java для основних функцій сайту: реєстрація, вхід, створення та відправка листівок, додавання е-гіфта.
- Розроблене програмне забезпечення автоматизованого димного тестування суттєво підвищило якість і надійність вебзастосунку, що дозволяє швидко реагувати на зміни та забезпечувати стабільну роботу сайту Givingli.
- Описано налаштування середовища розробки, створення нового проєкту в IntelliJ IDEA та налаштування структури директорій.
- Розроблено детальні інструкції по реалізації та запуску автотестів.

Таким чином, були виконання всі поставлені завдання. Димне тестування дозволило швидко оцінити основну працездатність системи та у виявляти критичні дефекти перед релізом кожної сборки. Автоматизовані тести забезпечили високу швидкість і ефективність перевірки, знижуючи ризик людських помилок.

10

Рисунок Б.10 – Слайд №10