

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 ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ АВТОМАТИЗАЦІЇ
ФУНКЦІОНАЛЬНОГО ТЕСТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ
SOFTWARE TO AUTOMATE FUNCTIONAL TESTING OF MOBILE
APPLICATIONS

Виконав(ла): студент(ка) 4 курсу, групи КНТ-210СП

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

ШАЧКОВ А.М.

(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»
 (повне найменування закладу вищої освіти)

Факультет ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 122 Комп'ютерні науки
 (код і найменування)
 Освітня програма (спеціалізація) Комп'ютерні науки
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2023 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ШАЧКОВА Артема Михайловича
 (ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення автоматизації функціонального тестування мобільного застосунку. Software to Automate Functional Testing of Mobile Applications

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

затверджені наказом закладу вищої освіти від “04” квітня 2023 року № 87

2. Строк подання студентом проєкту (роботи) 22 травня 2023 року

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження. 2. Матеріали і методи. 3. Опис програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

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

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ГЛАДОВА О.М., доцент		
Нормоконтроль	ДЕЙНЕГА Л.Ю., ст.викладач		

7. Дата видачі завдання « 04 » квітня 2023 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	2 тиждень	Розділ 2
	розробки.		
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження	5 тиждень	Розділ 4
	програмного забезпечення.		
7	Оформлення пояснювальної записки та документів	6 тиждень	Додатки
	до неї.		
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

_____ Артем ШАЧКОВ
(підпис) (Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

_____ Ольга ГЛАДКОВА
(підпис) (Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 67 с., 2 табл., 24 рис., 2 дод., 17 джерел.

АВТОТЕСТИ, ТЕСТОВІ СЦЕНАРІЇ, АВТОМАТИЗАЦІЯ ТЕСТУВАННЯ МОБІЛЬНИХ ЗАСТОСУНКІВ, ANDROID STUDIO

Об'єкт дослідження – теоретичні аспекти, методи та інструменти, що застосовуються для автоматизації процесу тестування мобільних застосунків.

Предмет дослідження – автоматизація процесу тестування мобільних застосунків.

Мета роботи – розробка автотестів для автоматизації тестування мобільного застосунку.

У ході дослідження були проаналізовані наявні інструменти автоматизації тестування мобільних застосунків, їх переваги та недоліки, а також визначені оптимальні методи тестування, які можуть бути автоматизовані. Робота включає огляд методів тестування, таких як функціональне, регресійне, стрес-тестування та тестування продуктивності, а також дослідження можливостей їх автоматизації.

На основі проведених досліджень була розроблена стратегія автоматизації тестування мобільних застосунків, яка включає вибір оптимальних інструментів та методів, планування автоматизації, розробку автоматизованих тестів та їх підтримку в процесі розробки мобільних застосунків.

Висновки. Отже, дана дипломна робота є актуальною в області автоматизації тестування мобільних застосунків, розроблене програмне забезпечення сприяє оптимізації тестування, покращенню якості мобільних продуктів та розвитку нових підходів та технологій в цій сфері.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 67 pages, 2 tables, 24 figures, 2 appendixes, 17 sources.

AUTO TESTS, TEST SCENARIOS, AUTOMATION OF TESTING MOBILE APPLICATIONS, ANDROID STUDIO

The object of the research is theoretical aspects, methods and means of automating the process of testing mobile applications.

The subject of research is the automation of the mobile application testing process.

The purpose of the work is the development of autotests for the automation of mobile application testing.

In the course of the study, the available means of automating the testing of mobile applications were analyzed, their advantages and disadvantages were determined, as well as the optimal methods of testing that can be automated. The work includes an overview of testing methods, such as functional, regression, stress testing and performance testing, as well as exploring the possibilities of their automation.

Based on the conducted research, a strategy for the automation of mobile application testing has been developed, which includes the selection of optimal tools and methods, automation planning, the development of automated tests and their support in the process of developing mobile applications.

Conclusions. Therefore, this thesis is relevant in the field of automation of testing of mobile applications, the developed software contributes to the optimization of testing, improvement of the quality of mobile products and the development of new approaches and technologies in this field.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	8
Вступ.....	9
1 Теоретичні аспекти автоматизації тестування мобільних застосунків	10
1.1 Актуальність теми.....	10
1.2 Мета і завдання дослідження.....	10
1.3 Об'єкт та предмет дослідження	11
1.4 Методи дослідження.....	11
1.5 Основні поняття та принципи тестування мобільних застосунків	12
1.6 Види автоматизованого тестування мобільних застосунків	13
1.7 Особливості автоматизації тестування мобільних застосунків	14
1.8 Переваги та недоліки автоматизації тестування мобільних застосунків	15
2 Аналіз інструментів та методів автоматизації тестування мобільних застосунків	17
2.1 Огляд наявних інструментів автоматизації тестування мобільних застосунків	17
2.2 Вибір оптимальних інструментів для автоматизації тестування мобільних застосунків	18
2.3 Огляд методів автоматизації тестування мобільних застосунків	20
2.4 Вибір оптимальних метод	21
3 Розробка стратегії автоматизації тестування мобільних застосунків	24
3.1 Визначення цілей та завдань автоматизації тестування	24
3.2 Планування автоматизації тестування.....	24
3.3 Впровадження та моніторинг автоматизації тестування	25
3.4 Оцінка ефективності автоматизації тестування.....	25
3.5 Навчання та розвиток команди.....	26
4 Основні рішення з реалізації тестування автоматизованого мобільного застосунку	27

4.1 Обґрунтування вибору середовища для розробки тестів автоматизації мобільних застосунків	27
4.2 Appium Server GUI	27
4.3 Appium Inspector.....	28
4.4 Android Studio Espresso.....	29
5 Керівництво з встановленням і запуском програмного забезпечення	30
5.1 Встановлення Android Studio	30
5.2 Встановлення Appium Server GUI і Appium Inspector.....	30
5.3 Запуск програми.....	31
5.4 Опис скрипту «Test»	31
Висновки	39
Перелік джерел посилань	40
Додаток А Текст програми.....	42
Додаток Б Слайди презентації	62

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
UI	– User Interface;
IDE	– Integrated Development Environment,
UX	– User Experience;
OC	– Operating System;
CI/CD	– Continuous Integration / Continuous Deployment;
SDK	– Software Development Kit;
QA	– Quality Assurance;
HAXM	–Hardware Accelerated Execution Manager.

ВСТУП

У сучасному світі мобільні технології посідають важливе місце в житті більшості людей. Мобільні застосунки стають все більш різноманітними та функціональними, а їх роль в економіці, освіті, розвагах та інших сферах життя продовжує зростати. Разом з цим зростають вимоги до якості та стабільності мобільних продуктів, що робить процес тестування мобільних застосунків критично важливим.

Автоматизація тестування мобільних застосунків є одним з напрямків, які допомагають забезпечити високу якість та надійність мобільних продуктів. Автоматизовані тести дозволяють швидко та ефективно перевіряти функціональність, продуктивність та стабільність мобільних застосунків на різних платформах та пристроях. Однак, процес автоматизації тестування мобільних застосунків має свої особливості, які вимагають глибокого розуміння і врахування специфіки мобільних платформ та доступних технологій.

Метою даної дипломної роботи є дослідження теоретичних аспектів, методів та інструментів автоматизації тестування мобільних застосунків, а також розробка стратегії автоматизації тестування, яка враховує специфіку мобільних платформ, доступні технології та ресурси.

1 ТЕОРЕТИЧНІ АСПЕКТИ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ МОБІЛЬНИХ ЗАСТОСУНКІВ

1.1 Актуальність теми

У сучасному світі мобільні застосунки займають важливе місце у повсякденному житті людей. Зростаюча кількість мобільних пристроїв, різноманітність операційних систем та апаратного забезпечення вимагає від розробників створювати якісні, продуктивні та надійні програми. Одним із важливих аспектів розробки мобільних застосунків є тестування. Від його якості залежить успіх програмного продукту на ринку.

Автоматизація тестування мобільних застосунків стає все більш актуальною темою, оскільки дозволяє забезпечити високу продуктивність, ефективність та економію часу. Однак процес автоматизації тестування мобільних застосунків має свої особливості, пов'язані зі специфікою мобільних пристроїв та застосунків [1].

1.2 Мета і завдання дослідження

Метою даної дипломної роботи є дослідження теоретичних аспектів, методів та інструментів автоматизації тестування мобільних застосунків, а також розробка стратегії автоматизації тестування, яка враховує специфіку мобільних платформ, доступні технології та ресурси.

Для досягнення поставленої мети в роботі будуть вирішуватися наступні завдання:

- вивчити теоретичні аспекти автоматизації тестування мобільних застосунків, включаючи принципи, переваги та недоліки автоматизації;
- оглянути та проаналізувати наявні методи та інструменти автоматизації тестування мобільних застосунків, їх застосування, можливості та обмеження;

- вибрати оптимальні методи та інструменти для автоматизації тестування мобільних застосунків, з урахуванням їх ефективності, сумісності, наявності ресурсів та вимог до тестування;
- розробити програмне забезпечення для автоматизації тестування мобільних застосунків, яке включає підхід до планування, проектування, розробки та виконання автоматизованих тестів, а також моніторингу та оцінки результатів тестування;
- описати можливі напрямки розвитку автоматизації тестування мобільних застосунків, з урахуванням сучасних технологій та тенденцій розвитку мобільних платформ.

1.3 Об'єкт та предмет дослідження

Об'єкт дослідження - процес тестування мобільних застосунків.

Предмет дослідження - методи та інструменти автоматизації тестування мобільних застосунків.

Об'єкт та предмет дослідження взаємопов'язані і спрямовані на вивчення автоматизації тестування мобільних застосунків, а також на розробку та впровадження ефективної стратегії автоматизації тестування з використанням сучасних методів та інструментів. Результати дослідження можуть бути використані під час розробки та впровадження автоматизованого тестування для забезпечення якості мобільних застосунків.

1.4 Методи дослідження

У даній роботі використовуються наступні методи дослідження:

- аналітичний метод – вивчення наукової літератури, статей та інших джерел, що стосуються автоматизації тестування мобільних застосунків, для формування теоретичної бази дослідження;

- синтез – об'єднання різних ідей, концепцій та практичних рішень, що стосуються автоматизації тестування мобільних застосунків, для розробки оптимальної стратегії;
- компаративний аналіз – порівняння різних методів та інструментів автоматизації тестування мобільних застосунків для вибору найбільш підходящих та ефективних рішень;
- експериментальний метод – практична реалізація автоматизації тестування мобільних застосунків з використанням обраних методів та інструментів, що дозволяє перевірити ефективність розробленої стратегії;
- статистичний метод – збір, аналіз та інтерпретація даних, отриманих під час експериментального дослідження, для оцінки ефективності автоматизації тестування мобільних застосунків.

Ці методи допомагають отримати об'єктивні результати, які можуть бути використані для покращення процесу автоматизації тестування мобільних застосунків та забезпечення якості розробленого програмного продукту [2].

1.5 Основні поняття та принципи тестування мобільних застосунків

Тестування мобільних застосунків - це процес перевірки функціональності, продуктивності, безпеки та інших аспектів мобільного застосунка з метою забезпечення його відповідності вимогам користувачів та стандартам якості.

Основні принципи тестування мобільних застосунків:

- раннє тестування: проводити тестування на ранніх етапах розробки, щоб виявити та виправити помилки з мінімальними витратами часу та ресурсів;
- цілісний підхід: перевіряти всі аспекти мобільного застосунка, включаючи функціональність, продуктивність, безпеку, сумісність, користувацький інтерфейс та інше;

- тестування на різних пристроях: переконатися, що застосунок працює належним чином на різних типах мобільних пристроїв, з різними операційними системами, версіями та апаратним забезпеченням;
- автоматизація: використання автоматизованих інструментів та методів для підвищення швидкості та ефективності тестування.

1.6 Види автоматизованого тестування мобільних застосунків

Автоматизоване тестування мобільних застосунків використовує різні техніки та інструменти для автоматичного виконання тестів, що спрощує процес тестування та забезпечує більш стабільні та надійні результати. Основні види автоматизованого тестування мобільних застосунків включають:

- автоматизоване функціональне тестування: використання тестових скриптів та інструментів для автоматичного перевірки функціональності та відповідності специфікації мобільного застосунка;
- автоматизоване тестування інтерфейсу користувача (UI): використання інструментів для автоматичної перевірки відповідності елементів інтерфейсу користувача дизайну та вимогам зручності користування;
- автоматизоване тестування продуктивності: застосування інструментів та методик для автоматичного моніторингу та аналізу продуктивності застосунка під час виконання різних завдань;
- автоматизоване стрес-тестування: використання тестових інструментів та технік для створення високого навантаження на застосунок та виявлення можливих проблем з продуктивністю та стабільністю;
- автоматизоване тестування сумісності: використання інструментів для автоматичного тестування роботи застосунка на різних пристроях, платформах та версіях операційних систем;

- автоматизоване регресійне тестування: створення та виконання тестових сценаріїв для перевірки впливу змін коду на існуючу функціональність застосунка;
- автоматизоване тестування безпеки: використання інструментів та технік для автоматичного виявлення вразливостей та перевірки захищеності застосунка від різних видів загроз;
- автоматизоване тестування локалізації: використання інструментів для автоматичної перевірки адаптації застосунка до різних мов, культур та регіональних налаштувань;
- автоматизоване альфа- та бета-тестування: використання спеціальних платформ та інструментів для автоматизації процесу збору відгуків від реальних користувачів та виявлення можливих проблем, що не були виявлені під час попередніх етапів тестування;
- контроль якості коду (Code Quality Assurance): використання автоматизованих інструментів для аналізу коду, виявлення проблем зі структурою, синтаксисом та стилем кодування, які можуть впливати на якість та стабільність застосунка;
- неперервна інтеграція (Continuous Integration) та неперервне розгортання (Continuous Deployment): використання автоматизованих систем для забезпечення постійної інтеграції змін коду, автоматичного тестування та розгортання оновлень застосунка.

Використання автоматизованого тестування мобільних застосунків дозволяє розробникам швидше виявляти та виправляти проблеми, забезпечує більш високу якість продукту та зменшує витрати на тестування [3].

1.7 Особливості автоматизації тестування мобільних застосунків

Автоматизація тестування мобільних застосунків має свої особливості, пов'язані зі специфікою мобільних пристроїв та застосунків:

- різноманітність пристроїв: автоматизовані тести мають враховувати широкий спектр мобільних пристроїв з різним апаратним забезпеченням та розмірами екранів;
- операційні системи: мобільні застосунки можуть працювати на різних операційних системах (iOS, Android, Windows Phone), що вимагає підтримки різних платформ при автоматизації тестування;
- тестування користувацького інтерфейсу: автоматизація тестування має враховувати специфіку користувацького інтерфейсу мобільних застосунків та взаємодію з ним;
- інтеграція з іншими системами: мобільні застосунки можуть інтегруватися з різними системами (сервери, API, бази даних), що вимагає відповідного підходу до автоматизації тестування.

1.8 Переваги та недоліки автоматизації тестування мобільних застосунків

Переваги автоматизації тестування мобільних застосунків:

- підвищення швидкості тестування: автоматизовані тести можуть виконуватись значно швидше, ніж ручні, що дозволяє розробникам швидше виявляти та виправляти проблеми;
- повторюваність та консистентність: автоматизовані тести можуть виконуватись з однаковою точністю та послідовністю кожного разу, забезпечуючи надійні результати;
- зменшення помилок: автоматизація знижує ймовірність помилок, які можуть виникнути через людський фактор під час ручного тестування;
- ефективне використання ресурсів: автоматизовані тести дозволяють краще використовувати час та зусилля команди розробників, зосереджуючись на важливіших аспектах проекту;
- регресійне тестування: автоматизація дозволяє швидко проводити регресійне тестування після внесення змін у код, що забезпечує стабільність продукту;

- паралельне тестування: можливість одночасного тестування на кількох пристроях, платформах та версіях операційних систем.

Недоліки автоматизації тестування мобільних застосунків:

- високі витрати на початкову інвестицію: вартість інструментів автоматизації та налаштування тестового середовища може бути високою;

- складність підтримки тестів: автоматизовані тести можуть вимагати значних зусиль для підтримки та оновлення відповідно до змін у коді та вимогах до застосунка;

- обмеження в автоматизації: деякі аспекти мобільних застосунків можуть бути важко автоматизувати, особливо щодо взаємодії з користувачем та візуального оформлення;

- навчання персоналу: перехід на автоматизоване тестування може вимагати додаткового навчання для розробників та тестувальників, що може бути трудомістким та затратним;

- залежність від інструментів: вибір інструментів для автоматизації тестування може обмежувати можливості тестування, залежно від їхньої функціональності та сумісності з платформами та мовами програмування.

Незважаючи на деякі недоліки, автоматизація тестування мобільних застосунків може значно підвищити ефективність тестування, забезпечити більшу стабільність та якість продукту, а також знизити витрати на довготривалі проекти. Однак, варто пам'ятати, що автоматизація не повинна повністю замінювати ручне тестування, а слугувати доповненням до нього, щоб виявити та виправити найбільш критичні проблеми [4].

2 АНАЛІЗ ІНСТРУМЕНТІВ ТА МЕТОДІВ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ МОБІЛЬНИХ ЗАСТОСУНКІВ

2.1 Огляд наявних інструментів автоматизації тестування мобільних застосунків

На сьогоднішній день існує багато інструментів для автоматизації тестування мобільних застосунків. В таблиці 2.1 наводиться порівняння.

Таблиця 2.1 – Порівняння середовищ автоматизації мобільних застосунків

Критерії	Appium	Espresso	XCTest	Kobiton
Мови програмування	Java, Ruby, Python, PHP, C#, JavaScript	Java, Kotlin	Swift, Objective-C	Java, Ruby, Python, PHP, JavaScript, C#
Платформи	Android, IOS	Android	IOS	Android, IOS
Взаємодія з елементами UI	Взаємодіє з елементами через API WebDriver	Взаємодіє з елементами напряду JVM	Взаємодіє з елементами напряду в XCTest	Використовує Appium або XCTest для взаємодії з елементами
Тестування на різних версіях ОС	Підтримує різні версії ОС	Обмежено сумісність з версіями Android	Обмежено сумісність з версіями iOS	Підтримує різні версії ОС
Складність налаштування	Середня	Легка	Середня	Легка
Підтримка різних пристроїв	Може працювати з різними пристроями та емуляторами	Робота виключно з емуляторам и Android	Робота виключно з емуляторам и iOS	Може працювати з різними пристроями та емуляторами

Appium - відкритий інструмент для автоматизації тестування мобільних застосунків для Android, iOS та Windows. Appium підтримує різні мови програмування, такі як Java, Ruby, Python та інші [5].

Espresso (Android) [6] і XCTest (iOS) [7] - інструменти, розроблені Google та Apple відповідно для автоматизації тестування користувацького інтерфейсу на їх платформах.

Detox - інструмент для end-to-end тестування мобільних застосунків, розроблених на React Native, який підтримує iOS та Android [8].

Ranorex - комерційний інструмент для автоматизації тестування мобільних застосунків, що пропонує підтримку різних платформ, ефективне керування тестовими

Robot Framework - відкритий інструмент для автоматизації тестування, що пропонує багато плагінів для роботи з мобільними застосунками, підтримку різних мов програмування та легкість використання [9].

2.2 Вибір оптимальних інструментів для автоматизації тестування мобільних застосунків

При виборі інструментів для автоматизації тестування мобільних застосунків варто враховувати наступні критерії [10]:

- сумісність з платформами: інструмент має підтримувати тестування на різних платформах (Android, iOS, Windows Phone), що дозволить максимально охопити аудиторію користувачів;
- мови програмування: важливо, щоб інструмент підтримував мови програмування, з якими зручно працювати розробникам та тестувальникам;
- легкість використання та налаштування: інструмент має бути зручним у використанні, мати можливість легко інтегруватися з середовищем розробки та системами керування версіями;

- спільнота та підтримка: відкритість коду та активна спільнота користувачів і розробників можуть полегшити роботу з інструментом та вирішення можливих проблем;
- ціна: враховуйте вартість інструменту, особливо якщо він є комерційним. Порівняйте його можливості та вартість з альтернативами.

Таблиця 2.2 – Порівняння середовищ автоматизації мобільних застосунків

Критерії	TestNG	Detox	Ranorex	Robot Framework
Мови програмування	Java	JavaScript	.NET,C#,VB.NET	Python, Java, JavaScript
Платформи	Cross-platform	IOS, Android	Windows, IOS, Android	Cross-platform
Взаємодія з елементами UI	Взаємодіє з елементами через Selenium	Взаємодіє з елементами EarlGrey(IOS) Espresso (Android)	Взаємодіє з елементами Selenium, EarlGrey(IOS) Espresso (Android)	Взаємодіє з елементами Selenium
Тестування на різних версіях ОС	Підтримує різні версії ОС	Підтримує різні версії ОС	Підтримує різні версії ОС	Підтримує різні версії ОС
Складність налаштування	Середня	Складна	Середня	Легка
Підтримка різних пристроїв	Може працювати з різними пристроями та емуляторами	Може працювати з різними пристроями та емуляторами	Може працювати з різними пристроями та емуляторами	Може працювати з різними реальними пристроями та емуляторами

З урахуванням зазначених критеріїв, можна вибрати оптимальний інструмент для автоматизації тестування мобільних застосунків згідно з потребами вашої команди та проекту. Наприклад, Appium може бути хорошим вибором для крос-платформного тестування, в той час як Espresso та XCTest можуть бути підходящими для тестування нативних застосунків на Android та iOS відповідно.

2.3 Огляд методів автоматизації тестування мобільних застосунків

Автоматизація тестування мобільних застосунків включає в себе різні методи та підходи наведені нижче [11].

Тестування користувацького інтерфейсу (UI тестування): цей метод передбачає перевірку коректності взаємодії користувача з елементами інтерфейсу та правильність відображення елементів на різних екранах та пристроях. Зазвичай використовуються спеціалізовані інструменти, такі як Appium, Espresso (Android) та XCTest (iOS).

Функціональне тестування: метод спрямований на перевірку коректності роботи функцій та можливостей мобільного застосунку відповідно до вимог. Автоматизація функціонального тестування забезпечує повторюваність перевірок та зменшення ризику помилок.

End-to-end тестування: цей метод передбачає перевірку взаємодії мобільного застосунку з іншими системами, такими як сервери, API, бази даних тощо. За допомогою автоматизації end-to-end тестування можна перевірити сценарії в реальному середовищі користувача.

Процеси Continuous Integration (CI) та Continuous Deployment (CD): ці методи передбачають автоматизацію процесів збірки, тестування та розгортання мобільних застосунків. За допомогою CI/CD можна автоматично запускати тести при зміні коду та отримувати швидкий зворотній зв'язок щодо якості програмного продукту [9].

Стрес-тестування та тестування продуктивності: ці методи допомагають виявити проблеми з продуктивністю та стабільністю мобільного застосунку під високим навантаженням.

Автоматизоване тестування локалізації: метод, який передбачає написання тестів, що перевіряють правильність перекладу та адаптації мобільного застосунку для різних мов та культур.

Автоматизоване регресійне тестування: метод, який передбачає написання тестів для перевірки впливу змін у коді на існуючі функції мобільного застосунку. Це допомагає виявити нові помилки, які можуть виникнути в результаті внесення змін або виправлення існуючих помилок.

Автоматизоване тестування взаємодії між компонентами: метод, що передбачає написання тестів для перевірки інтеграції та взаємодії між різними компонентами мобільного застосунку, такими як сервери, бази даних та інші служби. Автоматизоване тестування користувацького досвіду (UX): метод, який передбачає написання тестів, що оцінюють користувацький досвід з мобільного застосунку, включаючи навігацію, доступність та відгук на взаємодію користувачів.

Автоматизоване тестування API: метод, який передбачає написання тестів для перевірки роботи API мобільного застосунку, включаючи коректність відповідей, обробку помилок та захист даних.

Кожен з цих методів автоматизації тестування може бути використаний у різних стадіях розробки мобільних застосунків та має свої особливості та вимоги. Вибір методів тестування для автоматизації залежить від специфіки проекту, доступних ресурсів та цілей розробки. Важливо ретельно проаналізувати потреби проекту та врахувати всі переваги та недоліки автоматизації перед її впровадженням [12].

2.4 Вибір оптимальних метод

При виборі оптимальних методів автоматизації тестування мобільних застосунків необхідно враховувати ряд факторів.

Цілі та потреби проекту: визначте ключові цілі тестування, такі як перевірка функціональності, коректність інтерфейсу, продуктивність, безпека тощо. Оберіть методи, які найкраще підходять для досягнення цих цілей.

Бюджет та ресурси: розгляньте наявність фінансових та людських ресурсів для впровадження автоматизації тестування. Виберіть методи та інструменти, які відповідають вашим бюджетним обмеженням і можуть бути успішно інтегровані в вашу робочу групу.

Технічні вимоги та обмеження: оцініть технічні особливості вашого мобільного застосунку, такі як підтримувані платформи, архітектура, мова програмування, зовнішні сервіси, з якими взаємодіє застосунок тощо. Оберіть методи та інструменти, які будуть сумісні з вашим проектом і забезпечать максимальну ефективність [13].

Рівень складності та масштабування: врахуйте складність вашого мобільного застосунку та можливість розширення тестової бази в майбутньому. Виберіть методи, які дозволять вам легко масштабувати автоматизацію тестування та адаптувати її до змін у проекті.

Час впровадження: оцініть, скільки часу потрібно для впровадження автоматизації тестування у вашому проекті. Виберіть методи, які дозволять вам максимально ефективно реалізувати автоматизацію тестування, забезпечуючи оптимальний баланс між швидкістю впровадження та якістю тестування.

Підтримка та розвиток: зверніть увагу на рівень підтримки та розвитку обраних методів тестування та відповідних інструментів. Важливо, щоб інструменти та методи мали активну спільноту користувачів, регулярні оновлення та можливість адаптації під зміни в технологіях та стандартах [14].

Після аналізу цих факторів та з урахуванням специфіки вашого мобільного застосунку, вам слід обрати комбінацію методів автоматизації тестування, яка найкраще відповідає вашим потребам. При цьому можна використовувати як один метод, так і комбінацію кількох методів для досягнення оптимального результату тестування.

Оптимальний вибір методів автоматизації тестування мобільних застосунків забезпечить високу якість продукту, швидкість виявлення та вирішення проблем, а також зниження витрат на тестування та підтримку мобільного застосунку в довгостроковій перспективі.

3 РОЗРОБКА СТРАТЕГІЇ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ МОБІЛЬНИХ ЗАСТОСУНКІВ

3.1 Визначення цілей та завдань автоматизації тестування

Першим кроком у розробці стратегії автоматизації тестування мобільних застосунків є визначення цілей та завдань. Цілі мають бути конкретними, вимірюваними, досяжними, релевантними та обмеженими у час. Завдання повинні бути спрямовані на досягнення цілей.

Деякі приклади цілей та завдань автоматизації тестування можуть включати:

- покращення якості продукту: забезпечити відсутність критичних помилок, зменшити кількість регресій та забезпечити відповідність вимогам;
- зменшення часу тестування: скоротити час, потрібний для виконання тестів, та підвищити продуктивність команди;
- забезпечення безпеки продукту: перевірити застосунок на потенційні вразливості та недоліки з точки зору безпеки;
- оптимізація ресурсів: зниження витрат на тестування та підтримку продукту.

3.2 Планування автоматизації тестування

Створити план автоматизації тестування, включаючи такі етапи, як [15]:

- аналіз тестових сценаріїв: визначте тестові сценарії, які будуть автоматизовані, на основі їхньої критичності, частоти виконання та складності;
- розробка тестових сценаріїв: напишіть тестові сценарії, які враховують різні ситуації та варіанти користувачів;
- розробка тестових скриптів: створіть автоматизовані тестові скрипти, використовуючи вибрані методи та інструменти;

- налаштування тестового середовища: налаштуйте тестове середовище, забезпечуючи правильні налаштування, залежності та ресурси;
- виконання автоматизованих тестів: запустіть автоматизовані тести, відстежуйте їхній прогрес та аналізуйте результати;
- оцінка результатів тестування: перегляньте результати тестів, виявіть проблеми та вирішуйте їх, вносячи відповідні зміни в застосунок;
- підтримка та оновлення тестових скриптів: забезпечте регулярне оновлення тестових скриптів та сценаріїв у відповідь на зміни в мобільному застосунку та вимогах до нього.

3.3 Впровадження та моніторинг автоматизації тестування

Впровадження автоматизації тестування мобільних застосунків вимагає інтеграції автоматизованих тестів у процес розробки та регулярного моніторингу результатів тестування. Розгляньте використання системи безперервної інтеграції (Continuous Integration) для автоматичного запуску тестів при зміні вихідного коду.

Встановлюйте процеси та практики для моніторингу результатів автоматизованих тестів, включаючи аналіз помилок, виявлення проблем з продуктом та вирішення цих проблем. Забезпечте регулярний зворотний зв'язок розробникам, щоб вони могли вчасно реагувати на проблеми, виявлені під час тестування.

3.4 Оцінка ефективності автоматизації тестування

Для визначення успіху стратегії автоматизації тестування важливо проводити регулярні оцінки ефективності. Використовуйте показники продуктивності та якості, такі як:

- кількість виявлених та вирішених дефектів;
- час, потрібний для виконання автоматизованих тестів;
- рівень покриття коду тестами;

- витрати на тестування та підтримку продукту.

Ці показники допоможуть вам визначити, чи досягаються цілі автоматизації тестування, та виявити можливі вектори для поліпшення.

3.5 Навчання та розвиток команди

Розвиток навичок та знань команди є важливим аспектом успішної стратегії автоматизації тестування. Забезпечте, щоб члени команди мали доступ до необхідних ресурсів, навчання та підтримки для ефективного використання вибраних методів та інструментів автоматизації тестування.

Організуйте регулярні навчальні сесії, семінари або воркшопи для підвищення кваліфікації команди та забезпечення постійного розвитку їхніх навичок.

Розробка ефективної стратегії автоматизації тестування мобільних застосунків вимагає планування, вибору методів та інструментів, підтримки та моніторингу, а також розвитку команди. Реалізація цієї стратегії допоможе підвищити якість продукту, зменшити час тестування, оптимізувати витрати на тестування та підтримку продукту, а також забезпечити безпеку мобільного застосунку [16].

4 ОСНОВНІ РІШЕННЯ З РЕАЛІЗАЦІЇ ТЕСТУВАННЯ АВТОМАТИЗОВАНОГО МОБІЛЬНОГО ЗАСТОСУНКУ

4.1 Обґрунтування вибору середовища для розробки тестів автоматизації мобільних застосунків

При виборі середовища для автоматизації тестування мобільних застосунків важливо врахувати ряд факторів, які стосуються специфіки проекту, ресурсів.

Спочатку, Espresso було обрано як основний інструмент для автоматизації тестування Android застосунків. Espresso є частиною Android Studio, надає широкі можливості для тестування користувацького інтерфейсу та взаємодії з елементами. Додатково, використання Espresso дозволяє забезпечити високу продуктивність тестів та точність результатів, а також використовувати різноманітні мови програмування, зокрема Java і Kotlin.

Також, в процесі дослідження було зрозуміло, що для покращення ефективності процесу автоматизації, важливо використовувати мультиплатформні інструменти. Так, було вирішено використати Appium як сервер для автоматизації тестування. Appium підтримує роботу з різними платформами та мовами програмування, включаючи Java, що дозволяє йому ідеально інтегруватися з Espresso. Appium також має активну спільноту та є відкритим програмним забезпеченням, що забезпечує постійну підтримку та розвиток інструменту.

Отже, обрання комбінації Espresso та Appium як середовища для розробки тестів автоматизації мобільних застосунків зумовлено рядом переваг: широкі можливості для тестування UI, висока продуктивність, підтримка різних платформ і активна спільнота розробників.

4.2 Appium Server GUI

Appium Server GUI (графічний інтерфейс користувача) це інструмент, що дозволяє керувати сервером Appium за допомогою графічного

інтерфейсу. Це означає, що вам не потрібно вручну вводити команди в командний рядок або термінал для запуску або налаштування сервера Appium.

Використання Appium Server GUI може значно спростити процес автоматизації тестування, особливо для тих, хто не має глибоких знань з роботи в командному рядку. Він надає зручний інтерфейс для налаштування параметрів сервера, створення сесій, перегляду журналів і т.д.

Важливо відзначити, що для використання Appium Server GUI вам потрібно встановити його на ваш комп'ютер. Це окрема програма, яка використовує ту саму базу коду, що і Appium, але має додатковий графічний інтерфейс [5].

4.3 Appium Inspector

Appium Inspector є компонентом Appium, що служить інструментом для візуального аналізу мобільних застосунків. Його основна мета - допомагати розробникам і QA інженерам знайти елементи в інтерфейсі застосунків для автоматизованого тестування.

Ось особливості Appium Inspector:

- Appium Inspector забезпечує візуальне представлення ієрархії елементів UI в мобільному застосунку. Це полегшує визначення локаторів елементів, що потрібні для тестування;
- Appium Inspector може записувати взаємодію користувача з застосунком і автоматично генерувати відповідний код для автоматизованих тестів;
- за допомогою Appium Inspector ви можете відправляти команди до мобільного пристрою прямо з інтерфейсу інструменту, що дозволяє вам інтерактивно тестувати застосунок і відразу бачити результати [5].

4.4 Android Studio Espresso

Android Studio є основною інтегрованою середою розробки (IDE) для платформи Android, створеною Google. Вона містить в собі набір потужних інструментів, що полегшують процес розробки мобільних застосунків.

Espresso є частиною Android Studio та служить інструментом для автоматизації тестування UI мобільних застосунків. Espresso дозволяє розробникам створювати сценарії тестування, які імітують взаємодію користувача з мобільним застосунком, забезпечуючи при цьому велику надійність та стабільність тестів [6].

5 КЕРІВНИЦТВО З ВСТАНОВЛЕННЯМ І ЗАПУСКОМ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Встановлення Android Studio

Заходимо на офіційний сайт Android Developers і качаємо Android Studio та встановлюємо.

Відкриваємо Android Studio. Програма потребує встановити SDK для цього потрібно вибрати вільну теку і запам'ятати шлях для Appium Server GUI.

Також потрібно буде встановити HAXM (Intel Hardware Accelerated Execution Manager) для емулятора і з цим може бути проблема так як Android Studio не правильно встановлене.

Перше рішення – можна зайти папку де Android Studio скинула сам exe файл для встановлення HAXM.

Також може бути проблема якщо у біос Hardware Accelerated Execution Manager заборонено. Чи може бути що це AMD процесор.

5.2 Встановлення Appium Server GUI і Appium Inspector

Крок 1. Заходимо на офіційний Git програми так як це середовище не має свого сайту [17].

Крок 2. Справа знизу натисніть на кнопочку “Release” і скачайте середовище.

Крок 3. Коли скачали, встановіть Appium Server GUI і Appium Inspector.

Крок 4. Запусте Appium Server GUI вам потрібно клікнути на кнопку “Edit Configurations” і там вказати для “ANDROID_HOME” путь в SDK папку.

Крок 5. Appium Inspector нічого не потрібно робити поки ви не написали код для автоматизації.

5.3 Запуск програми

По перше запускаємо Appium Server GUI а також створюємо локальний сервер (LocalHost).

По друге запускаємо Android Studio і також емулятор і застосунок для якого писався тест.

Запускаєм скрипт “Test” після чого спостерігаємо за процесом.

5.4 Опис скрипту «Test»

Всього було розроблено та проведено 4 тести.

Перший тест містить в себе тестування активності кнопки на відповідний колір.

Що робить тест: У вкладці “Main menu” натискає на кнопку “Catalog”.
Що очікує тест: Після переходу с вкладки “Main menu” активна кнопка повинна змінити колір на не активну, а кнопка “Catalog” повинна стати активною.

Після виконання: зображено приклад успішного виконання першого тесту на рисунку 5.1. Коли всі завдання виконані успішно відповідна інформація відображається у консолі (рис. 5.2).

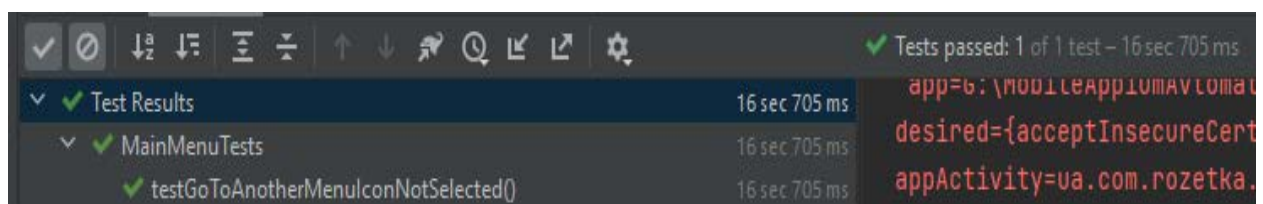


Рисунок 5.1 – Тест пройдено успішно

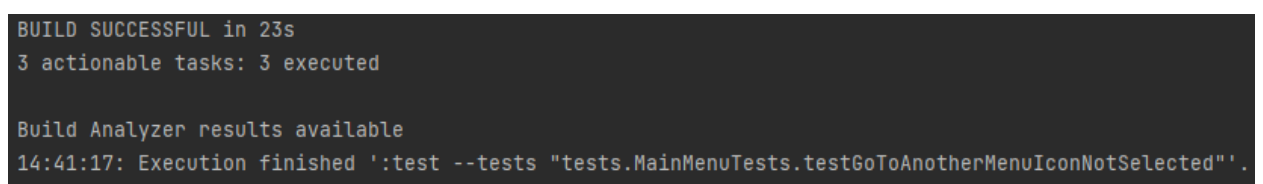


Рисунок 5.2 – Тест виконав усі завдання.

Діаграма послідовності виконання першого тесту зображена на рисунку 5.3.



Рисунок 5.3 – Діаграма послідовності роботи першого тесту

Другий тест містить в собі перевірку знаходження, додавання товару в корзину, а також перевірку назви товару, чи правильно добавило кількість товару.

Що робить тест: На вкладці “Main menu” натискає на кнопку пошук, знаходить товар, відкриває вкладку товару, додає в кошик товар, переходить на вкладку кошик, перевіряє назву товару які він додав, а також кількість товару які він додав.

Що очікує тест: Що товар буде доданий в корзину однією штукаю і після переходу побачить доданий їм товар з правильною назвою і кількістю.

Після виконання: зображено приклад успішного виконання другого тесту на рисунку 5.4. Коли всі завдання виконані успішно відповідна інформація відображається у консолі (рис. 5.5).

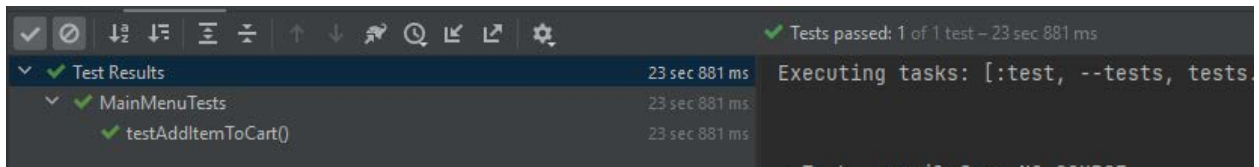


Рисунок 5.4 – Другий тест пройдено успішно

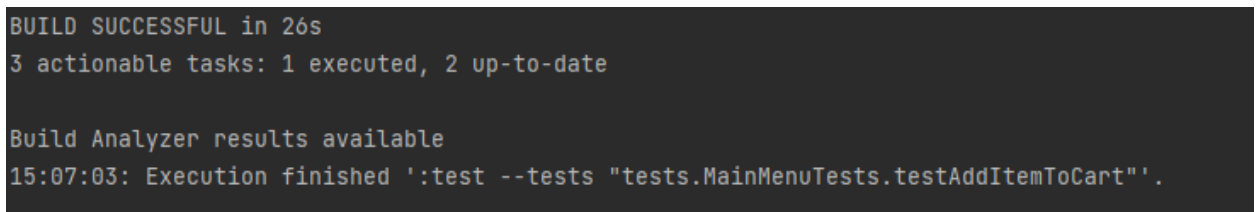


Рисунок 5.5 – Другий тест виконав усі завдання.

Діаграма послідовності виконання другого тесту зображена на рисунку 5.6.

Третій тест містить в себе фотографію повного екрану застосунку і порівнює розмір екрану телефону.

Що робить тест: У вкладці “Main menu” переходить до вкладки “Catalog”, фотографує екран.

Що очікує тест: При порівнянні фотографія та екран повинні бути однакові. Якщо тест виконано з помилкою то з’являється в консолі про це повідомлення (рис. 5.7).

Що насправді трапляється: При порівнянні фотографія виявляється меншою за екран.

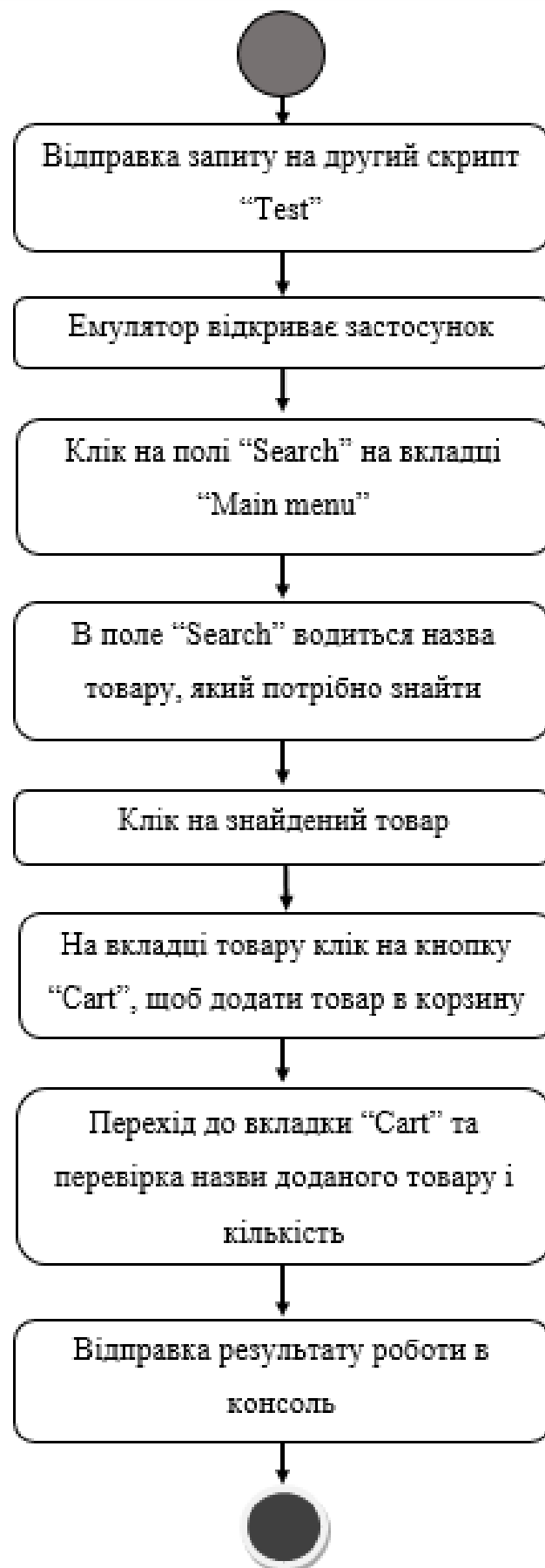


Рисунок 5.6 – Діаграма послідовності роботи другого тесту

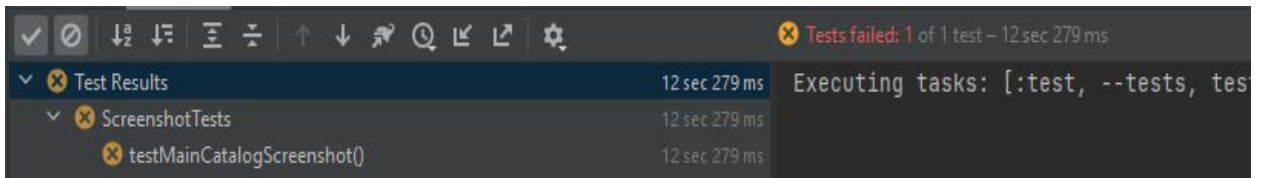


Рисунок 5.7 – Третій тест не пройдено успішно

Після виконання: зображено приклад не успішного виконання третього тесту на рисунку 5.7. На рисунку 5.8 зображено пояснення проблеми яка є в програмі після того коли тест завершився.

Коли всі завдання виконані успішно відповідна інформація відображається у консолі (рис. 5.9).

```
expected: <MATCH> but was: <SIZE_MISMATCH>
Expected :MATCH
Actual   :SIZE_MISMATCH
```

Рисунок 5.8 – Третій тест показує що очікуваний і насправді не однакові

```
BUILD FAILED in 14s
3 actionable tasks: 1 executed, 2 up-to-date
15:17:49: Execution finished ':test --tests "tests.ScreenshotTests.testMainCatalogScreenshot"'.
```

Рисунок 5.9 – Третій тест виконав усі завдання

Діаграма послідовності виконання третього тесту зображена на рисунку 5.10.

Четвертий тест містить в себе порівняння фотографії сторінки “Main menu” і екрану “Main menu” але ми порівняння з фотографією попереднього тесту тому він буде фейлитися.

Що очікує тест: То що екран і фотографія будуть однакові з однаковими об’єктами. Результат помилки в консолі зображено на рисунку 5.11.



Рисунок 5.10 – Діаграма послідовності роботи третього тесту

Що насправді трапляється: То що фотографія береться з минулого тесту і порівнюється з фотографією з вкладки “Main menu”.

Після виконання: зображено приклад не успішного виконання четвертого тесту на рисунку 5.11. На рисунку 5.12 зображено пояснення проблеми яка є в програмі після того коли тест завершився.

Коли всі задачі виконані успішно інформація відображається у консолі (рис. 5.13).

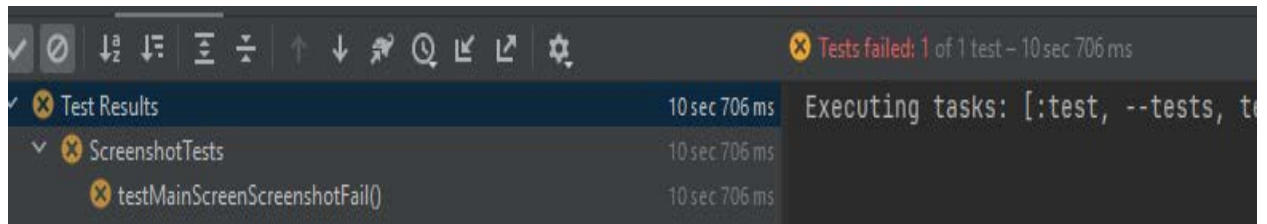


Рисунок 5.11 – Четвертий тест не пройдено успішно

```
expected: <MATCH> but was: <SIZE_MISMATCH>
Expected :MATCH
Actual   :SIZE_MISMATCH
```

Рисунок 5.12 – Четвертий тест показує що очікуваний і насправді не
одинакові

```
BUILD FAILED in 13s
3 actionable tasks: 1 executed, 2 up-to-date
15:32:10: Execution finished ':test --tests "tests.ScreenshotTests.testMainScreenScreenshotFail"'.

```

Рисунок 5.13 – Четвертий тест виконав усі завдання.

Діаграма послідовності виконання четвертого тесту зображена на рисунку 5.14.



Рисунок 5.14– Діаграма послідовності роботи четвертого тесту

ВИСНОВКИ

Автоматизація тестування мобільних застосунків є важливою складовою успішного розроблення та підтримки мобільних продуктів. Вона допомагає поліпшити якість та стабільність застосунків, скоротити час на тестування та виявити проблеми на ранніх етапах розробки.

У дипломній роботі були розглянуті теоретичні аспекти автоматизації тестування мобільних застосунків, аналіз інструментів та методів автоматизації, а також розроблена стратегія автоматизації тестування. Основні етапи розробки стратегії включають визначення цілей та завдань автоматизації тестування, вибір методів та інструментів, планування автоматизації, впровадження та моніторинг автоматизованих тестів, оцінку ефективності автоматизації та розвиток команди.

Реалізація ефективної стратегії автоматизації тестування мобільних застосунків може сприяти успіху розробки мобільних продуктів, забезпечуючи високу якість, стабільність та безпеку застосунків на ринку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Mobile Labs. Mobile Labs: Mobile Device Cloud [Electronic resource]. – Access mode: <https://www.mobilelabsinc.com>.
2. Fewster, M. Software Test Automation: Effective Use of Test Execution Tools / M.Fewster, D.Graham. – Addison-Wesley Professional, 1999. – 596 p.
3. Dustin, E. Automated Software Testing: Introduction, Management, and Performance / E.Dustin, J.Rashka, J.Paul. – Addison-Wesley Professional, 1999. – 608 p.
4. Myers, G. J. The Art of Software Testing / G.J.Myers, C.Sandler, T.Badgett. – John Wiley & Sons, 2011. – 256 p.
5. Appium. Appium: Mobile App Automation Made Awesome [Electronic resource]. – Access mode: <http://appium.io>.
6. Google. Espresso: Write Android UI tests [Electronic resource]. – Access mode: <https://developer.android.com/training/testing/espresso>.
7. Apple Inc. XCTest: Test iOS apps [Electronic resource]. – Access mode: <https://developer.apple.com/documentation/xctest>.
8. Robot Framework: A generic open source automation framework [Electronic resource]. – Access mode: <https://robotframework.org>.
9. Ranorex Studio: Accelerate testing with a full-featured IDE [Electronic resource]. – Access mode: <https://www.ranorex.com/ranorex-studio/>.
10. Dzone. Mobile Testing: Challenges, Strategies, and Best Practices [Electronic resource]. – Access mode: <https://dzone.com/articles/mobile-testing-challenges-strategies-and-best-prac>.
11. Kobiton. Kobiton: Mobile Device Testing Platform [Electronic resource]. – Access mode: <https://www.kobiton.com>.
12. Perfecto. Perfecto: Continuous Testing Platform [Electronic resource]. – Access mode: <https://www.perfecto.io>.
13. ISTQB. International Software Testing Qualifications Board [Electronic resource]. – Access mode: <https://www.istqb.org>.

14. Jenkins. Jenkins: Open Source Automation Server [Electronic resource]. – Access mode: <https://www.jenkins.io>.
15. JUnit. JUnit: A programmer-oriented testing framework for Java [Electronic resource]. – Access mode: <https://junit.org>.
16. TestNG. TestNG: A testing framework for the JVM [Electronic resource]. – Access mode: <https://testng.org/doc>.
17. Appium Inspector [Electronic resource]. – Access mode: <https://github.com/appium/appium-inspector>.

ДОДАТОК А
Текст програми

```

plugins {
    id 'java'
    id "io.qameta.allure" version "2.9.6"
}

group 'org.example'
version '1.0-SNAPSHOT'

def allureVersion = "2.19.0",
    junitVersion = "5.8.2",
    slf4jVersion = "1.7.32"

repositories {
    mavenCentral()
}

dependencies {
    testImplementation 'com.github.romankh3:image-comparison:4.4.0'
    testImplementation(
        "io.qameta.allure:allure-junit5:$allureVersion",
        "com.codeborne:selenide:5.25.0",
        "io.qameta.allure:allure-selenide:$allureVersion",
        "io.appium:java-client:7.6.0",
        "org.aeonbits.owner:owner:1.0.12",
        "io.rest-assured:rest-assured:4.4.0",
        "org.junit.jupiter:junit-jupiter-api:$junitVersion")
    testRuntimeOnly(
        "org.slf4j:slf4j-simple:$slf4jVersion",
        "org.junit.jupiter:junit-jupiter-engine:$junitVersion")
}

test {
    useJUnitPlatform()

    testLogging {
        lifecycle {
            events "failed"
            exceptionFormat "short"
        }
    }
}

```

```
}
```

```
package config;
```

```
import org.aeonbits.owner.ConfigFactory;
```

```
public class ConfigReader {
```

```
    public static final EmulatorConfig emulatorConfig =
    ConfigFactory.create(EmulatorConfig.class, System.getProperties());
    public static final TestConfig testConfig =
    ConfigFactory.create(TestConfig.class, System.getProperties());
}
```

```
@Config.LoadPolicy(Config.LoadType.MERGE)
@Config.Sources({
    "system:properties",
    "file:src/test/resources/configs/emulator.properties",
})
```

```
public interface EmulatorConfig extends Config {
    @Key("deviceName")
    String deviceName();
```

```
    @Key("platformName")
    String platformName();
```

```
    @Key("appPackage")
    String appPackage();
```

```
    @Key("appActivity")
    String appActivity();
```

```
    @Key("app")
    String app();
```

```
    @Key("remoteURL")
    String remoteURL();
```

```
}
```

```

package config;

import org.aeonbits.owner.Config;

@Config.LoadPolicy(Config.LoadType.MERGE)
@Config.Sources({
    "system:properties",
    "file:src/test/resources/configs/test.properties",
})
public interface TestConfig extends Config {
    @Key("updateScreenshots")
    @DefaultValue("false")
    boolean isScreenshotsNeedToUpdate();

    @Key("deviceHost")
    String deviceHost();
}

package driver;

import com.codeborne.selenide.WebDriverProvider;

import config.ConfigReader;
import helper.ApkInfoHelper;
import io.appium.java_client.android.AndroidDriver;

import io.appium.java_client.remote.AndroidMobileCapabilityType;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.remote.DesiredCapabilities;

import javax.annotation.Nonnull;

import java.io.File;
import java.net.MalformedURLException;
import java.net.URL;

import static org.junit.jupiter.api.Assertions.assertTrue;

public class EmulatorDriver implements WebDriverProvider {
    protected static AndroidDriver driver;
    private static final String DEVICE_NAME =

```

```

ConfigReader.emulatorConfig.deviceName();
    private static final String PLATFORM_NAME =
ConfigReader.emulatorConfig.platformName();
    private static String APP_PACKAGE =
ConfigReader.emulatorConfig.appPackage();
    private static String APP_ACTIVITY =
ConfigReader.emulatorConfig.appActivity();
    private static final String APP = ConfigReader.emulatorConfig.app();
    private static final String URL = ConfigReader.emulatorConfig.remoteURL();

    public static URL getUrl() {
        try {
            return new URL(URL);
        } catch (MalformedURLException e) {
            throw new RuntimeException(e);
        }
    }

    private String getAbsolutePath(String filePath) {
        File file = new File(filePath);
        assertTrue(file.exists(), filePath + " not found");

        return file.getAbsolutePath();
    }

    private void initPackageAndActivity() {
        ApkInfoHelper helper = new ApkInfoHelper();
        APP_PACKAGE = APP_PACKAGE.isEmpty() ?
helper.getAppPackageFromApk() : APP_PACKAGE;
        APP_ACTIVITY = APP_ACTIVITY.isEmpty() ? helper.getAppMainActivity() :
APP_ACTIVITY;
    }

    @NonNull
    @Override
    public WebDriver createDriver(DesiredCapabilities desiredCapabilities) {
        initPackageAndActivity();

        desiredCapabilities.setCapability(AndroidMobileCapabilityType.AUTO_GRANT_
PERMISSIONS,true);
        desiredCapabilities.setCapability("deviceName", DEVICE_NAME);
        desiredCapabilities.setCapability("platformName", PLATFORM_NAME);

        desiredCapabilities.setCapability("appPackage", APP_PACKAGE);
        desiredCapabilities.setCapability("appActivity", APP_ACTIVITY);
    }

```

```

        desiredCapabilities.setCapability("app", getAbsolutePath(APP));
        driver = new AndroidDriver<>(getUrl(), desiredCapabilities);
        return driver;
    }
}

```

```
package driver;
```

```

import com.codeborne.selenide.SelenideElement;
import io.appium.java_client.android.nativekey.AndroidKey;
import io.appium.java_client.android.nativekey.KeyEvent;

```

```
public class EmulatorHelper extends EmulatorDriver {
```

```

    public static void goBack() {
        driver.navigate().back();
    }

```

```

    public static void androidScrollToAnElementByText(String text) {
        driver.findElementByAndroidUIAutomator("new UiScrollable(new
        UiSelector().scrollable(true)" +
            ".instance(0)).scrollIntoView(new UiSelector().textContains(\"" + text +
            "\").instance(0))")
            .click();
    }

```

```

    public static void closeKeyBoard() {
        if (driver.isKeyboardShown()) {
            driver.hideKeyboard();
        }
    }

```

```

    public static void sendKeysAndFind(SelenideElement element, String text) {
        element.sendKeys(text);
        driver.pressKey(new KeyEvent(AndroidKey.ENTER));
    }

```

```
}
```

```
package helper;
```

```
import config.ConfigReader;
```

```
import java.io.IOException;
```

```
import java.util.concurrent.ExecutionException;
```

```
import java.util.regex.Matcher;
```

```
import java.util.regex.Pattern;
```

```
import static helper.DeviceHelper.*;
```

```
public class ApkInfoHelper {
    private String apkInfo;
```

```
    public ApkInfoHelper() {
        String app = ConfigReader.emulatorConfig.app(); //читаем путь к арк из
        проптерей
        if(app == null || app.isEmpty()){ //если путь к арк файлу не указан,
        выкидываем ошибку
            throw new RuntimeException("No value for key 'app' providing apk path in
            emulator.properties");
        }
        try {
            String pathToAapt = "G:\\LOCATION SDK\\build-tools\\33.0.2\\";
            apkInfo = executeSh("aapt dumb badging " +
            ConfigReader.emulatorConfig.app());
        } catch (IOException | InterruptedException | ExecutionException e){
            throw new RuntimeException(e);
        }
    }
}
```

```
public String getAppPackageFromApk() {
    return findGroup1ValueFromString(apkInfo,"package: name=\"\\s*([^\"]+?)\\s*\"");
}
```

```
public String getAppMainActivity() {
    return findGroup1ValueFromString(apkInfo, "launchable-activity:
    name=\"\\s*([^\"]+?)\\s*\"");
}
```

```

    }

    private static String findGroup1ValueFromString(String text, String regex){
        Pattern pattern = Pattern.compile(regex);
        Matcher matcher = pattern.matcher(text);
        if(matcher.find()){
            return matcher.group(1);
        }
        return null;
    }
}

package helper;
public class Constants {
    public static String SCREENSHOT_TO_SAVE_FOLDER = "screenshots/actual/";
}

package helper;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.FutureTask;
import java.util.stream.Collectors;

public class DeviceHelper {

    public static String executeSh(String command) throws IOException,
    ExecutionException, InterruptedException {
        Process p = Runtime.getRuntime().exec(command);
        FutureTask<String > future = new FutureTask<>(()->{
            return new BufferedReader(new InputStreamReader(p.getInputStream()))
                .lines().map(Object::toString)
                .collect(Collectors.joining("\n"));
        });
        new Thread(future).start();
        return future.get();
    }

    public static String executeBash(String command) {

```

```

Process p;
try {
    p = Runtime.getRuntime().exec(command);
} catch (IOException e){
    throw new RuntimeException(e);
}
final String[] message = {""];

new Thread()->{
    BufferedReader input = new BufferedReader(new
InputStreamReader(p.getInputStream()));
    String line = null;
    while (true){
        try {
            if ((line = input.readLine()) == null) {
                break;
            }
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
        message[0] += line + "\n";
    }
    System.out.println(message[0]);
}).start();
try {
    p.waitFor();
} catch (InterruptedException e){
    throw new RuntimeException(e);
}
return message[0];
}
}

```

```
package helper;
```

```
import config.ConfigReader;
import driver.EmulatorDriver;
```

```
public class RunHelper {
```

```
    private RunHelper() {
    }

```

```

public static RunHelper runHelper() {
    return new RunHelper();
}

public Class<?> getDriverClass() {
    String deviceHost = ConfigReader.testConfig.deviceHost();

    switch (deviceHost) {
        case "browserstack":
        case "selenium":
        case "emulator":
            return EmulatorDriver.class;
        case "real":
        default:
            throw new RuntimeException("В файле конфигурации нет параметра
deviceHost: " +
                "browserstack/selenium/emulator/real");
    }
}
}

```

```

package listeners;

```

```

import com.codeborne.selenide.Selenide;
import io.qameta.allure.Attachment;
import org.junit.jupiter.api.extension.AfterTestExecutionCallback;
import org.junit.jupiter.api.extension.ExtensionContext;
import org.openqa.selenium.OutputType;

```

```

import java.lang.reflect.Method;

```

```

public class AllureListener implements AfterTestExecutionCallback {

```

```

    @Attachment(value = "Page screenshot", type = "image/png")
    public static byte[] saveScreenshot(byte[] screenShot) {
        return screenShot;
    }

```

```

    @Override
    public void afterTestExecution(ExtensionContext context) {
        Method testMethod = context.getRequiredTestMethod();
        String testName = testMethod.getName();
    }

```

```

        boolean testFailed = context.getExecutionException().isPresent();
        if (testFailed) {
            if (!testName.contains("Screenshot")) {
                saveScreenshot(Selenide.screenshot(OutputType.BYTES));
            }
        }
    }
}

package pages;

import static com.codeborne.selenide.Selenide.$;

import java.io.File;

import io.appium.java_client.MobileBy;

public class BasePage {

    public File fullPageScreenshot(){
        return $(MobileBy.className("android.widget.LinearLayout")).screenshot();
    }
}

import io.appium.java_client.MobileBy;
import io.qameta.allure.Step;

public class CartPage extends BasePage {

    private SelenideElement cartMenuIcon = $(MobileBy.id("iv_cart"));
    private SelenideElement cartMenuAfter = $(MobileBy.id("tab_cart"));

    private SelenideElement cartNumbers =
$(MobileBy.id("view_number_picker_et_value"));

    @Step
    public CartPage openCartPage(){
        cartMenuIcon.should(Condition.visible).click();
        cartMenuAfter.should(Condition.selected);
        return this;
    }
}

```

```

@Step
public CartPage assertItemCount(Integer count){
    String itemCountText =cartNumbers.getAttribute("text");

    Assertions.assertTrue( itemCountText != null &&
itemCountText.contains(String.valueOf(count)));
    return this;
}

package pages;

import static com.codeborne.selenide.Selenide.$;

import com.codeborne.selenide.Condition;
import com.codeborne.selenide.SelenideElement;

import io.appium.java_client.MobileBy;
import io.qameta.allure.Step;

public class CatalogPage extends SearchPage {

    private SelenideElement catalogMenuIcon = $(MobileBy.id("tab_fat_menu"));

    @Step
    public void isMainMenuIconSelected(){
        catalogMenuIcon.should(Condition.selected);
    }
}

package pages;

import static com.codeborne.selenide.Selenide.$;

import com.codeborne.selenide.Condition;
import com.codeborne.selenide.SelenideElement;

import org.openqa.selenium.By;

import io.appium.java_client.MobileBy;
import io.qameta.allure.Step;

```

```

public class ItemPage extends SearchPage {

    private SelenideElement addToCartBtn = $(MobileBy.id("iv_cart"));

    @Step
    public ItemPage addToCart(){
        addToCartBtn.should(Condition.visible).click();
        return this;
    }

    @Step
    public CartPage openCart(){
        return new CartPage().openCartPage();
    }
}

package pages;

import static com.codeborne.selenide.Selenide.$;
import static com.codeborne.selenide.Selenide.$$;
import static driver.EmulatorHelper.androidScrollToAnElementByText;
import static driver.EmulatorHelper.goBack;

import com.codeborne.selenide.CollectionCondition;
import com.codeborne.selenide.Condition;
import com.codeborne.selenide.ElementsCollection;
import com.codeborne.selenide.SelenideElement;
import com.codeborne.selenide.WebDriverRunner;

import java.util.List;

import io.appium.java_client.MobileBy;
import io.qameta.allure.Step;

public class MainPage extends SearchPage {

    private SelenideElement catalogButton = $(MobileBy.id("tab_fat_menu"));
    private ElementsCollection categories =
        $$$(MobileBy.id("block_offer_a_section"));
    private SelenideElement mainMenuIcon = $(MobileBy.id("tab_home"));
    private SelenideElement adPopUp = $(MobileBy.id("commonBtnSA"));

```

```

@Step
public CatalogPage openCatalog() {
    catalogButton.should(Condition.visible).click();
    return new CatalogPage();
}

```

```

@Step
public MainPage closePopUp() {
    if (adPopUp.exists()) {
        goBack();
    }
    return this;
}

```

```

@Step
public MainPage checkCategoriesHasTexts(List<String> texts) {
    androidScrollToAnElementByText(texts.get(0));
    categories.should(CollectionCondition.size(6));
    categories.should(CollectionCondition.textsInAnyOrder(texts));
    return this;
}

```

```

@Step
public void checkMainMenuButtonIsSelected() {
    mainMenuIcon.should(Condition.selected);
}

```

```

@Step
public void checkMainMenuButtonIsNotSelected() {
    mainMenuIcon.should(Condition.not(Condition.selected));
}
}

```

```
package pages;
```

```
import static com.codeborne.selenide.Selenide.$;
```

```

import static driver.EmulatorHelper.closeKeyBoard;
import static driver.EmulatorHelper.sendKeysAndFind;
import com.codeborne.selenide.Condition;
import com.codeborne.selenide.SelenideElement;

```

```

import driver.EmulatorHelper;
import io.appium.java_client.MobileBy;

```

```
import io.qameta.allure.Step;
```

```
public class SearchPage extends BasePage {
    private SelenideElement searchField = $(MobileBy.id("view_search_tv"));
    private SelenideElement searchFieldExpanded =
$(MobileBy.id("search_et_query"));
    private SelenideElement searchGlass = $(MobileBy.id("item_search_iv_left"));

    private SelenideElement searchNameON =
$(MobileBy.id("item_search_tv_title"));
    private SelenideElement tempItem;
```

```
@Step
public SearchPage search(String item) {
    searchField.should(Condition.visible).click();
    sendKeysAndFind(searchFieldExpanded, item);
    searchGlass.should(Condition.visible).click();
    searchNameON.should(Condition.visible).click();
    closeKeyBoard();
    return this;
}
```

```
@Step
public String getFirstFoundedFullItemName() {
    tempItem =
$(MobileBy.xpath("(//android.view.ViewGroup[4]//android.widget.TextView[@cl
ass=\"android.widget.TextView\"])[1]"));
    return tempItem.getText();
}
```

```
@Step
public ItemPage openFirstFoundItemByTextContains() {
    tempItem.click();
    return new ItemPage();
}
```

```
}
```

```
package tests;
```

```
import com.codeborne.selenide.Configuration;
import com.codeborne.selenide.Selenide;
import com.codeborne.selenide.logevents.SelenideLogger;
import com.github.romankh3.image.comparison.ImageComparison;
import com.github.romankh3.image.comparison.ImageComparisonUtil;
import com.github.romankh3.image.comparison.model.ImageComparisonResult;
import com.github.romankh3.image.comparison.model.ImageComparisonState;
```

```
import config.ConfigReader;
import io.qameta.allure.Allure;
import io.qameta.allure.selenide.AllureSelenide;
import listeners.AllureListener;
```

```
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.extension.ExtendWith;
```

```
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
```

```
import java.nio.file.StandardCopyOption;
```

```
import static helper.Constants.SCREENSHOT_TO_SAVE_FOLDER;
import static helper.DeviceHelper.executeBash;
import static helper.RunHelper.runHelper;
import static io.qameta.allure.Allure.step;
import static org.junit.jupiter.api.Assertions.assertEquals;
```

```
@ExtendWith(AllureListener.class)
public class BaseTest {
```

```
    @BeforeAll
    public static void setup() {
```

```
        SelenideLogger.addListener("AllureSelenide", new AllureSelenide());
```

```
        Configuration.reportsFolder = SCREENSHOT_TO_SAVE_FOLDER;
```

```

Configuration.browser = runHelper().getDriverClass().getName();
Configuration.startMaximized = false;
Configuration.browserSize = null;
Configuration.timeout = 10000;
disableAnimationOnEmulator();
}

```

```

private static void disableAnimationOnEmulator() {
    executeBash("adb -s shell settings put global transition_animation_scale 0.0");
    executeBash("adb -s shell settings put global window_animation_scale 0.0");
    executeBash("adb -s shell settings put global animator_duration_scale 0.0");
}

```

```

public void assertScreenshot(File actualScreenshot, String expectedFileName) {

```

```

    expectedFileName = expectedFileName.replace("()", ".png");

```

```

    String expectedScreensDir = "src/test/resources/expectedScreenshots/";

```

```

    if (ConfigReader.testConfig.isScreenshotsNeedToUpdate()) {
        try {

```

```

            Files.move(actualScreenshot.toPath(), new File(expectedScreensDir +
expectedFileName).toPath(),
                StandardCopyOption.REPLACE_EXISTING);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }

```

```

        return;
    }

```

```

    BufferedImage expectedImage = ImageComparisonUtil
        .readImageFromResources(expectedScreensDir + expectedFileName);

```

```

    BufferedImage actualImage = ImageComparisonUtil
        .readImageFromResources(SCREENSHOT_TO_SAVE_FOLDER +
actualScreenshot.getName());

```

```

    File resultDestination = new File("diff/diff_" + expectedFileName);

```

```

    ImageComparisonResult imageComparisonResult = new
ImageComparison(expectedImage, actualImage, resultDestination)
    .compareImages();

    if
(imageComparisonResult.getImageComparisonState().equals(ImageComparisonSt
ate.MISMATCH)) {
        try {

            byte[] diffImageBytes = Files.readAllBytes(resultDestination.toPath());
            AllureListener.saveScreenshot(diffImageBytes);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }

    }

    assertEquals(ImageComparisonState.MATCH,
imageComparisonResult.getImageComparisonState());
}

@BeforeEach
public void startDriver() {
    step("Відкрити додаток", (Allure.ThrowableRunnableVoid) Selenide::open);
}

@AfterEach
public void afterEach() {
    step("Закрити додаток", Selenide::closeWebDriver);
}

}

package tests;

import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;

import java.util.Arrays;
import java.util.List;

```

```

import io.qameta.allure.Description;
import pages.MainPage;
import pages.SearchPage;

public class MainMenuTests extends BaseTest {

    private static MainPage mainPage;

    @BeforeAll
    public static void init() {
        mainPage = new MainPage();
    }

    @Test
    public void testGoToAnotherMenuIconNotSelected(){
        mainPage.closePopUp();
        mainPage.checkMainMenuButtonIsSelected();
        mainPage.openCatalog().isMainMenuIconSelected();
        mainPage.checkMainMenuButtonIsNotSelected();
    }

    @Test
    public void testAddItemToCart(){
        String searchItem = "processor";
        SearchPage searchPage = mainPage.search(searchItem);
        String fullName = searchPage.getFirstFoundedFullItemName();

        searchPage.openFirstFoundItemByTextContains()
            .addToCart()
            .openCart()
            .assertItemsCount(1)
            .assertItemNameInCart(fullName);
    }
}

package tests;

import pages.MainPage;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.TestInfo;

```

```

import java.io.File;

public class ScreenshotTests extends BaseTest{
    private TestInfo testInfo;
    @BeforeEach
    public void init(TestInfo testInfo) {
        this.testInfo = testInfo;
    }

    @Test
    public void testMainCatalogScreenshot(){
        File mainScreenScreenshot = new MainPage().closePopUp()
            .openCatalog()
            .fullPageScreenshot();
        assertScreenshot(mainScreenScreenshot, testInfo.getDisplayName());
    }

    @Test
    public void testMainScreenScreenshotFail(){
        File mainScreenScreenshot = new MainPage()
            .fullPageScreenshot();
        assertScreenshot(mainScreenScreenshot, "testMainCatalogScreenshot()");
    }
}

deviceName=Pixel 4 API 33
platformName=android
version=13.0
appPackage=
appActivity=
remoteURL=http://127.0.0.1:4808/wd/hub
app=src/test/resources/apk/rozetka.apk

updateScreenshots=false
deviceHost=emulator

```

ДОДАТОК Б
Слайди презентації

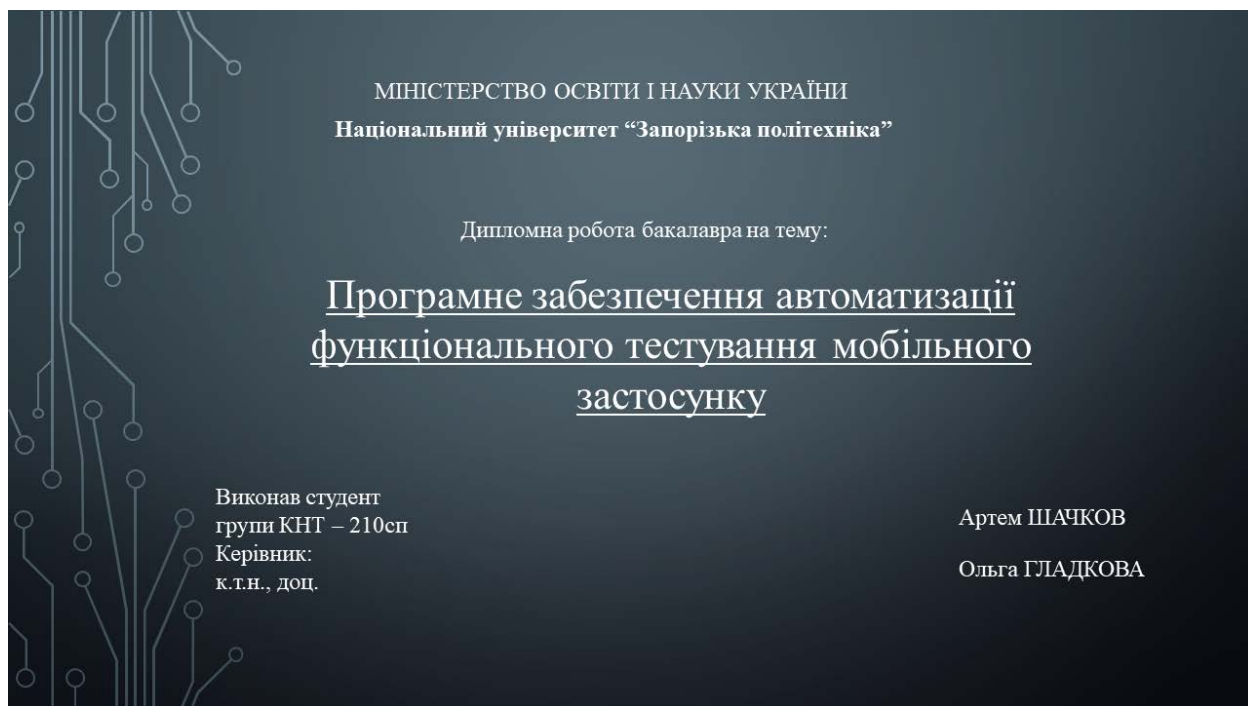


Рисунок Б.1 – Титульний слайд

Актуальність		
Види	Преваги	Недоліки
Автоматизація тестування Web застосунків	1. Можливість тестування в різних браузерях та версіях. Легкість інтеграції з різними системами. Велике співтовариство та багато навчальних ресурсів. Зазвичай включає в себе глибоке тестування функціональності.	1. Можливість ускладнень з перехідністю між браузерами. Потреба в постійному оновленні тестів для нових версій браузерів. Вища складність автоматизації при тестуванні розподілених систем.
Автоматизація тестування мобільних застосунків	1. Можливість тестування на різних платформах та пристроях. Можливість тестування реальних умов користування. Врахування специфіки мобільних ОС і апаратного забезпечення. Можливість тестування offline сценаріїв.	1. Складність в налаштуванні тестового середовища. Потреба в додатковому апаратному забезпеченні або емуляторах. Можливість ускладнень при адаптації до нових версій мобільних ОС.

Рисунок Б.2 – Актуальність

Мета і завдання дослідження

Метою даної дипломної роботи є розробка автотестів для автоматизації тестування мобільного застосунку.

Для досягнення поставленої мети в роботі будуть вирішуватися наступні завдання:

- дослідити теоретичні аспекти автоматизації тестування мобільних застосунків, включаючи принципи, переваги та недоліки автоматизації;
- оглянути та проаналізувати наявні методи та інструменти автоматизації тестування мобільних застосунків, їх застосування, можливості та обмеження;
- вибрати оптимальні методи та інструменти для автоматизації тестування мобільних застосунків, з урахуванням їх ефективності, сумісності, наявності ресурсів та вимог до тестування;
- розробити програмне забезпечення для автоматизації тестування мобільних застосунків, яке включає планування, проектування, розробки та виконання автоматизованих тестів, а також моніторингу та оцінки результатів тестування.

Рисунок Б.3 – Мета і завдання дослідження

Порівняльне дослідження середовищ розробки

Критерії	Appium	Espresso	XCTest	Kobiton
Мови програмування	Java, Ruby, Python, PHP, C#, JavaScript	Java, Kotlin	Swift, Objective-C	Java, Ruby, Python, PHP, JavaScript, C#
Платформи	Android, iOS	Android	iOS	Android, iOS
Взаємодія з елементами UI	Взаємодія з елементами через API WebDriver	Взаємодія з елементами напряму JVM	Взаємодія з елементами напряму в XCTest	Використовує Appium або XCTest для взаємодії з елементами
Тестування на різних версіях ОС	Підтримує різні версії ОС	Обмежено сумісність з версіями Android	Обмежено сумісність з версіями iOS	Підтримує різні версії ОС
Складність налаштування	Середня	Легка	Середня	Легка
Підтримка різних пристроїв	Може працювати з різними пристроями та емуляторами	Робота виключно з емуляторами Android	Робота виключно з емуляторами iOS	Може працювати з різними реальними пристроями та емуляторами

Рисунок Б.4 – Порівняльне дослідження середовищ розробки

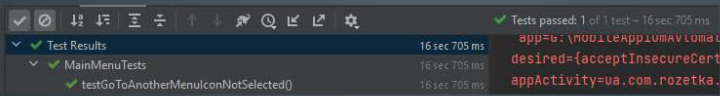
Перший тест

Що містить тест: перевіряє переключення активності кнопки і його переходу на не активну

Що робить тест: На вкладці “Main menu” натискає на кнопку “Catalog”.

Результат тесту

Перший тест пройшов успішно



Перший тест виконав усі завдання.

```

BUILD SUCCESSFUL in 23s
3 actionable tasks: 3 executed

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

Рисунок Б.5 – Опис Першого тесту

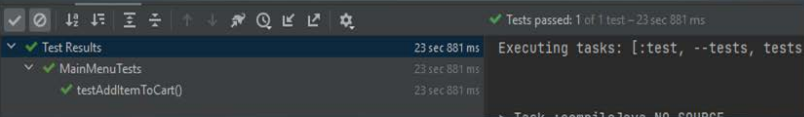
Опис Другого тесту

Що містить тест: перевірку знаходження, додавання, назви, кількість товару.

Що робить тест: натискає на кнопку пошук, знаходить товар, відкриває вкладку товару, додає до кошику товар, знаходить на сторінці кошик, перевіряє назву товару та кількість.

Результат тесту

Другий тест пройшов успішно



Другий тест виконав усі завдання.

```

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 Б.6 – Опис Другого тесту

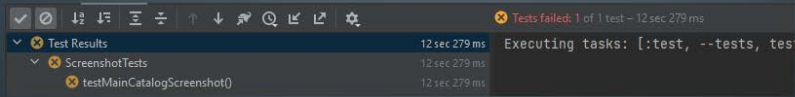
Опис Третього тесту

Що містить тест: Фотографування екрану додатку, зрівняння з екраном телефону

Що робить тест: З вкладки “Main menu” переходить на вкладку “Catalog”, робить скріншот екрану.

Результат тесту

Третій тест не пройшов успішно



Причина через що тест не пройшов успішно

```
expected: <MATCH> but was: <SIZE_MISMATCH>
Expected :MATCH
Actual   :SIZE_MISMATCH
```

Третій тест виконав усі завдання.

```
3 actionable tasks: 1 executed, 2 up-to-date
15:17:49: Execution finished ':test --tests "tests.ScreenshotTests.testMainCatalogScreenshot"'.
```

Рисунок Б.7 – Опис Третього тесту

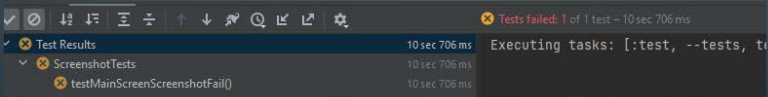
Опис Четвертого тесту

Що містить тест: Скріншот екрану зостасунку, порівняння з фотографією з третього тесту

Що робить тест: На вкладці “Main menu” робить скріншот екрану.

Результат тесту

Четвертий тест не пройшов успішно



Причина із за чого тест не пройшов успішно

```
expected: <MATCH> but was: <SIZE_MISMATCH>
Expected :MATCH
Actual   :SIZE_MISMATCH
```

Четвертий тест виконав усі завдання.

```
3 actionable tasks: 1 executed, 2 up-to-date
15:32:10: Execution finished ':test --tests "tests.ScreenshotTests.testMainScreenScreenshotFail"'.
```

Рисунок Б.8 – Опис Четвертого тесту

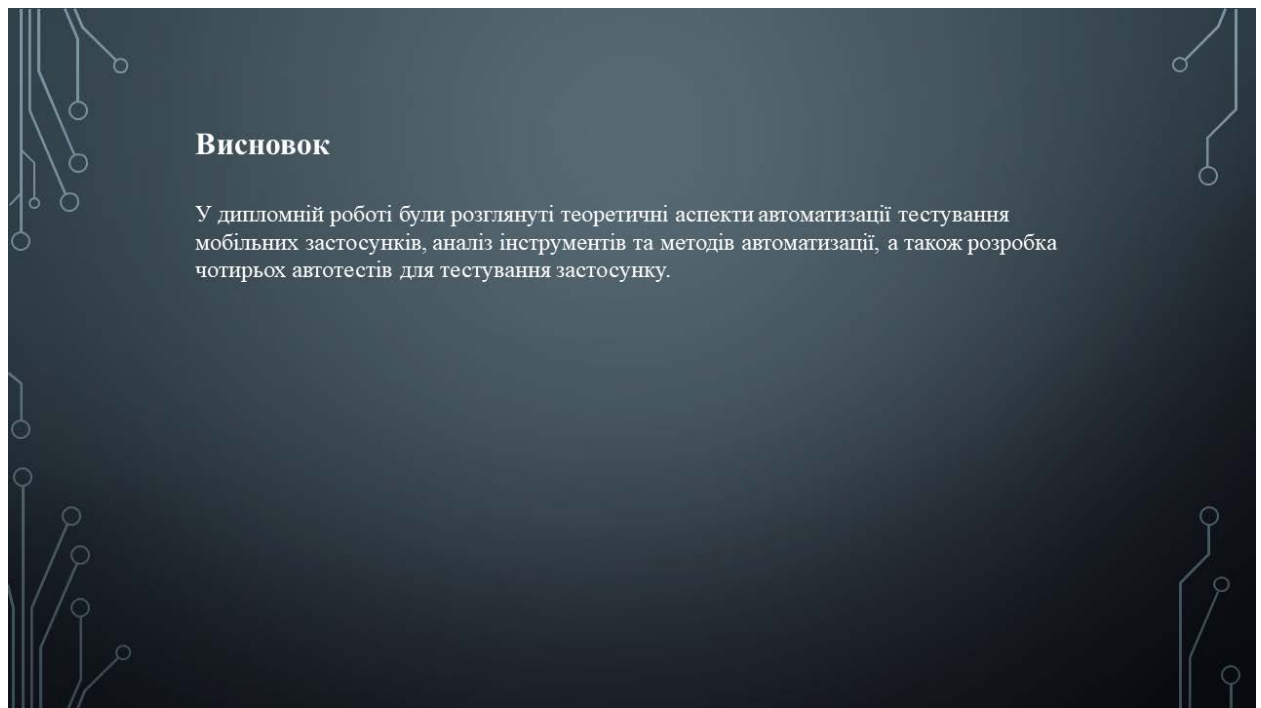


Рисунок Б.9 - Висновок