МІНІСТЕРСТВО ФІНАНСІВ УКРАЇНИ
ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ ФІНАНСІВ ТА ЦИФРОВИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА
на тему:

СИСТЕМА ГАРАНТУВАННЯ ЯКОСТІ МОБІЛЬНИХ ДОДАТКІВ

Здобувача вищої освіти першого (бакалаврського) рівня,
4 курсу, групи КБз-19-1, спеціальності 122 «Комп'ютерні науки»,
освітня програма «Інформаційні управляючі системи і технології (за галузями)»

_____ Гринюк Юлії Миколаївни
(підпис) (прізвище, ім'я, по батькові)

Науковий керівник _____ Гладченко Оксана Вікторівна, к.пед.н., доцент
(підпис) (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Керівник освітньо-професійної програми _____ Федотова - Півень Ірина Миколаївна, к.т.н, доцент
(підпис) (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту за рішенням кафедри
протокол від «01» червня 2023 № 14

Завідувач кафедри _____ Філоненко Михайло Миколайович, к.ф.-м.н., доцент
(підпис) (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Ірпінь – 2023

Реєстрація № 35 06.06.2023 _____
(дата) (підпис відповідальної особи)

Результати перевірки До захисту 01.06.2023 _____
(до захисту, на доопрацювання) (дата) (підпис)

Результат захисту 5 (відмінно) 16.06.2023 _____
(оцінка) (дата) (підпис)

Голова комісії: _____ В.М. Франчук
(підпис)

Члени комісії: _____ М.М. Філоненко
(підпис)

_____ В.О. Ніжегородцев
(підпис)

_____ Є.В. Котух
(підпис)

БЛАНК ОЦІНЮВАННЯ ВИПУСКНОЇ РОБОТИ

Об'єкти контролю	Діапазон оцінки в балах	Фактична кількість балів	Примітки	Дата, П.І.Б.: керівника, голови комісії членів комісії,секретаря, (підписи)
ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ КОМІСІЄЮ ПО ЗАХИСТУ				
1. Зміст випускної роботи (якість теоретичного матеріалу, його зв'язок з практичним, вміння представляти матеріал в логічній послідовності, наявність висновків та пропозицій)	від 0 до 30	29		16.06.2023 Франчук В.М. Філоненко М.М Ніжегородцев В.О. Котух Є.В. Суховецька Л.П.
2. Дотримання вимог щодо оформлення	від 0 до 10	8		
3. Вчасне виконання роботи на всіх етапах у встановлені кафедрою терміни	від 0 до 5	4		
4. Рецензент	від 0 до 5	4		
РАЗОМ БАЛІВ	від 0 до 50	45		

Оцінювання кваліфікаційної роботи науковим керівником - 46 балів.

ПЕРЕВЕДЕННЯ ОЦІНКИ ЗА 100-БАЛЬНОЮ ШКАЛОЮ ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалоюЄКТС	Оцінка за національною шкалою	Дата	П.І.Б.: голови, членів комісії, секретаря, (підпис)
91	A	5 (відмінно)	16.06.2023	Франчук В.М. Філоненко М.М Ніжегородцев В.О. Котух Є.В. Суховецька Л.П.

ПРАВИЛА ПЕРЕВЕДЕННЯ ОЦІНОК ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою
90-100	A	5 (відмінно)
80-89	B	4 (добре)
70-79	C	
60-69	D	3 (задовільно)
50-59	E	
35-49	FX	2 (незадовільно)

Факультет фінансів та цифрових технологій
Кафедра комп'ютерних та інформаційних технологій і систем
Галузь знань: 12 «Інформаційні технології»
Спеціальність: 122 «Комп'ютерні науки»
Освітня програма: «Інформаційні управляючі системи і технології (за галузями)»

Затверджую
Завідувач кафедри
_____ Філоненко М.М.
«16» 11 2022

Завдання на кваліфікаційну роботу
здобувачу вищої освіти
Гринюк Юлії Миколаївні
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи
Система гарантування якості мобільних додатків
Затверджена наказом ректора від «15» листопада 2022 № 1205 (зі змінами наказ від «04» квітня 2023 № 361)
2. Строк здачі закінченої роботи «01» червня 2023
3. Цільова установка провести дослідження та створення системи гарантування якості мобільного додатка
Мета роботи розробка автоматизованих тестів, щоб забезпечити ефективність тестування мобільного додатку
Об'єкт дослідження процес створення системи гарантування якості
Предмет дослідження засоби та інструменти, що використовуються для реалізації системи автоматизованого тестування
База дослідження мобільний додаток Udemy
4. Вихідні дані до роботи інформація про тестування, структура автоматизованого тестування, інструменти для автоматизації, мова програмування Python, середовище розробки
5. Перелік графічного матеріалу рисунки: ефект ерори, життєвий цикл програмного забезпечення, життєвий цикл тестування, рівні тестування, види та методи тестування, проста архітектура автоматизації тестування, загальна архітектура автоматизації тестування, логотип та частина коду мови програмування Python, як працює Appium на Android, скріншоти виконання програми; таблиці: порівняння та 2 тестові випадки; презентація
6. Зміст випускної кваліфікаційної роботи (перелік питань за кожним розділом)
Вступ, Розділ 1. Аспекти сучасних підходів та методів тестування, 1.1.Загальне поняття про тестування програмного забезпечення, 1.2.Вхідні та вихідні дані, 1.3.Мета тестування, 1.4.Рівні та види тестування, 1.5.Ручне та автоматизоване тестування, 1.6.Тестування мобільних додатків, Розділ 2. Архітектура автоматизованого тестування та засоби автоматизації, 2.1.Архітектура

автоматизованого тестування, 2.2.Мова програмування Python, 2.3.Інте середовище розробки, 2.4.Фреймворк Pytest, 2.5.Інструмент для автома Selenium, 2.6.Інструмент автоматизації Appium, 2.7.Система управління т TestRail, Розділ 3. Реалізація системи тестування, 3.1.Інформація про мо додаток Udemy, 3.2.Тестова документація, 3.3.Системні вимоги, 3.4.Процес р системи тестування, 3.5.Використання програмної системи, 3.6.Аналіз отр результатів, 3.7.Удосконалення автоматизованого тестування та гарантування якості в цілому, ВИСНОВКИ.

7. Календарний план виконання роботи

№ п/п	Назва етапів кваліфікаційноїроботи	Строк виконання етапів роботи	
		За планом	Фактично
1	2	3	4
1	Вибір теми, підбір літературних джерел, формування структури кваліфікаційної роботи	До 15.11.2022	15.11.2022
2	**І розділ** (розкрити сутність і стан дослідження наукової проблематики роботи, після чого навести її загальну характеристику; огляд літератури за темою і вибір конкретних напрямів досліджень)	06.02.2023	06.02.2023
3	**ІІ розділ** (аналіз експериментальної частини про методики досліджень)	31.03.2023	31.03.2023
4	**ІІІ розділ** (проведені теоретичні і (або) експериментальні дослідження та проектні рішення)	05.05.2023	17.04.2023
5	**Висновки**	16.05.2022	01.05.2022
6	Доопрацювання роботи з урахуванням зауважень наукового керівника	01.06.2023	01.06.2023
7	Отримання відгуку керівника	08.06.2023	08.06.2023
8	Отримання зовнішньої рецензії	11.06.2023	09.06.2023

8. Дата видачі завдання «16» листопада 2022

9. Керівник випускної кваліфікаційної роботи _____ (підпис) Гладченко О.Е (прізвище, ініціали)

10. Керівник освітньо-професійної програми _____ (підпис) Федотова-Пів (прізвище, ініціали)

11. Завдання до виконання отримав здобувач вищої освіти _____ (підпис) Гринюк Ю.М. (прізвище, ініціали)

АНОТАЦІЯ

Система гарантування якості мобільних додатків

Мета роботи полягає у розробці автоматизованих тестів, щоб забезпечити ефективність тестування мобільного додатку.

Завдання роботи включає дослідження методів тестування та необхідність тестування у життєвому циклі розробки програмного забезпечення, питання важливості саме автоматизованого тестування та певних інструментів автоматизації, які потрібні на практиці, створення системи гарантування якості та удосконалення автоматизації.

Предметом дослідження є засоби та інструменти, що використовуються для реалізації системи автоматизованого тестування.

Об'єктом дослідження є процес створення системи гарантування якості.

Базою дослідження є мобільний додаток Udemy.

За результатами дослідження розроблена система гарантування якості мобільного додатка.

Одержані результати можуть бути використані на будь-якому підприємстві, яке має на меті покращити якість розробленого мобільного додатку і тим самим підвищити задоволення користувачів та конкурентоспроможність.

Кваліфікаційна робота містить 72 сторінки, 7 таблиць, 18 рисунків, список використаних джерел з 25 найменувань, 3 додатки.

Ключові слова: автоматизоване тестування, інструменти для автоматизації, гарантування якості, тест-кейси, Python, Android SDK, Appium, емулятор.

Рік виконання кваліфікаційної роботи: 2023 р.

Рік захисту роботи: 2023 р.

ABSTRACT

Quality Assurance System for Mobile Applications

The aim of the work is to develop automated tests to ensure the efficiency of testing a mobile application.

The tasks of the work include researching testing methods and the necessity of testing in the software development life cycle, discussing the importance of automated testing and specific automation tools needed in practice, creating a quality assurance system, and improving automation.

The subject of the research is the tools and instruments used for implementing an automated testing system.

The object of the research is the process of creating a quality assurance system.

The research is based on the Udemy mobile application.

As a result of the research, a quality assurance system for the mobile application has been developed.

The obtained results can be used by any company aiming to improve the quality of their developed mobile application and, in turn, enhance user satisfaction and competitiveness.

The qualification paper consists of 72 pages, 7 tables, 18 figures, a list of 25 references, 3 appendices.

Keywords: automated testing, automation tools, quality assurance, test cases, Python, Android SDK, Appium, emulator.

Year of completion of the qualification paper: 2023.

Year of defending the paper: 2023.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ,СКОРОЧЕНЬ, ТЕРМІНІВ

OpenSource – відкритий код

Error – невірна дія програми, яка призводить до некоректного результату.

Defect – дефект, недолік програми, який знижує її якість.

IDE (Integrated Development Environment) – це комплексне середовище розробки програмного забезпечення, яке містить в собі різні інструменти, такі як редактор коду, компілятор, дебагер, підсвітку синтаксису та інші.

IDLE (Integrated Development and Learning Environment) – це інтерактивне середовище розробки Python, яке включає в себе редактор коду та інтерактивний інтерпретатор.

VCS (Version Control System) – програмне забезпечення, яке дозволяє зберігати, відстежувати та керувати версіями файлів та коду програм.

URL-адреса (або просто URL) – це унікальний ідентифікатор, що використовується для вказівки місцезнаходження ресурсу в Інтернеті.

USB-драйвер – це програмне забезпечення, яке дозволяє взаємодіяти з USB-пристроями на операційній системі.

UIAutomator – це фреймворк для тестування Android-додатків, розроблений компанією Google.

ADB (Android Debug Bridge) – це інструмент, що надає доступ до різноманітних функцій інженерії на пристроях з операційною системою Android.

TCP-сервер – це програмне забезпечення, що запускається на комп'ютері або сервері та надає можливість здійснювати з'єднання з іншими пристроями за допомогою протоколу передачі даних TCP/IP.

CI/CD (Continuous Integration/Continuous Delivery) – це практика розробки програмного забезпечення, яка полягає в автоматизації процесу збирання, тестування та розгортання програмного забезпечення.

ЗМІСТ

ВСТУП **11**

РОЗДІЛ 1. АСПЕКТИ СУЧАСНИХ ПІДХОДІВ ТА МЕТОДІВ ТЕСТУВАННЯ .. **14**

1.1. Загальне поняття про тестування програмного забезпечення 14

1.2. Вхідні та вихідні дані 19

1.3. Мета тестування 21

1.4. Рівні та види тестування 22

1.5. Ручне та автоматизоване тестування 27

1.6. Тестування мобільних додатків 28

1.6.1. Типи мобільних додатків 29

1.6.2. Прилади для тестування мобільних додатків 31

1.6.3. Виклики при тестуванні мобільних додатків 32

РОЗДІЛ 2. АРХІТЕКТУРА АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ТА ЗАСОБИ АВТОМАТИЗАЦІЇ **34**

2.1. Архітектура автоматизованого тестування 34

2.2. Мова програмування Python 39

2.3. Інтегроване середовище розробки 41

2.3.1. Pycharm 43

2.3.2. Android SDK 43

2.4. Фреймворк Pytest 46

2.5. Інструмент для автоматизації Selenium 47

2.6. Інструмент автоматизації Appium 48

2.7. Система управління тестами TestRail 53

РОЗДІЛ 3. РЕАЛІЗАЦІЯ СИСТЕМИ ТЕСТУВАННЯ **54**

3.1. Інформація про мобільний додаток Udemy 54

3.2. Тестова документація 54

3.3. Системні вимоги 59
3.4. Процес розробки системи тестування 60
3.5. Використання програмної системи 63
3.6. Аналіз отриманих результатів 65
3.7. Удосконалення автоматизованого тестування та системи гарантування якості в цілому 66
ВИСНОВКИ **68**
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ **70**
ДОДАТКИ **72**

ВСТУП

З переходом до цифрового суспільства сьогодні вплив програмного забезпечення набагато більший. Це сприяє підвищенню стандартів життя, покращує перспективи бізнесу та розвиває розумний світ. Окрім промислового/комерційного впливу, сьогодні програмне забезпечення відіграє вирішальну роль у всіх сферах життя звичайних людей. Однак усе, що для цього потрібно, – це безперебійна та безвідмовна робота програмного забезпечення.

Оскільки катастрофічні наслідки збоїв програмного забезпечення руйнують бізнес і соціальне населення, розглянемо, чому тестування програмного забезпечення та забезпечення якості є важливими для будь-якого бізнесу.

Непоправні збитки, які може спричинити збій програмного забезпечення, очевидні з вражаючих звітів по всьому світу. У 2017 році звіт, опублікований Tricentis, показує, що глобальні підприємства засвідчили втрату активів на суму 1,7 трильйона, що стосується 3,7 мільйона людей у всьому світі. У 2018 році в США низька якість програмного забезпечення призвела до жахливих втрат у сумі 2,84 трильйона доларів США.

Крім того, страшні історії підприємств, які постраждали від збоїв у програмному забезпеченні, які їх майже вибили, облетіли новинами по всьому світу. IT-проблеми, які засмутили різні бізнес-сектори, включаючи British Airways, Facebook, мережу O2 тощо, були оприлюднені як страхітливі наслідки збою програмного забезпечення.

Деякі компанії створюють власні відділи контролю якості та розвивають їх; це часто тривалий і громіздкий процес, який потребує значних витрат часу та грошей. (Рис. А.1., Додаток А)

Простіше кажучи, важливість тестування програмного забезпечення можна простежити з відповіді користувача. Це гарантує якість продукту та задовольняє клієнтів, а також користувачів. Крім того, це обіцяє кращу оптимізацію бізнесу

(менше витрат на обслуговування), надійність і чудову взаємодію з користувачем. По суті, ітераційні зусилля, витрачені на формування потужного, але безпомилкового програмного забезпечення, є дуже широкими [13].

Попри дизайнерів і розробників, тестування програмного забезпечення відіграє вирішальну роль у цьому процесі. Оскільки процес розробки слідує за життєвим циклом розробки програмного забезпечення, процес тестування стоїть відразу за чергою для компіляції блоків для перевірки та підтвердження.

Загалом тестування програмного забезпечення – це безперервний процес перевірки та валідації для надання чудового продукту. Кожна фаза тестування визначає продуктивність і ефективність системи або додатку. У результаті це допомагає переконатися, що програмне забезпечення відповідає всім технічним і бізнес-параметрам.

У цю цифрову чи технологічну еру ми оточені різним програмним забезпеченням, кількість якого з кожним днем стрімко зростає. Незалежно від того, яку операційну систему чи платформу застосовано, ми використовуємо програмне забезпечення та полегшуємо собі життя. Без програмного забезпечення комп'ютер чи смартфон – це не що інше, як набір різних апаратних компонентів.

Існують різні типи програмного забезпечення. Проте є два основних типи, на основі яких подаються інші типи ПЗ – це системне та прикладне програмне забезпечення.

Варто замітити: для мобільних додатків використовується саме прикладне програмне забезпечення. А нині, мобільні додатки важливі для ведення бізнесу. До того ж у 2022 році у світі налічувалось 6,65 мільярда користувачів смартфонів, що становить 86% світового населення. Смартфони – це не лише бурхлива індустрія, а й невід'ємна частина нашого повсякденного життя. Не дивно, що до 2022 року кількість завантажень мобільних додатків зросте до колосальних 258 мільярдів. Сьогодні існують додатки для будь-якого мобільного пристрою та призначення, і всі вони залежать від відгуків користувачів. Люди не пропустять нагоди залишити

поганий відгук, будь то проблеми з продуктивністю, поганий інтерфейс користувача чи збій, тому обов’язково потрібно інвестувати в тестування мобільного додатка [11]. І тому це означає, що мобільне тестування має бути невід’ємною частиною стратегії розвитку, щоб залишатися в конкурентному середовищі. Мобільне тестування гарантує, що програми та смартфони безпечні, надійні та зручні для користувачів (Рис. А.2., Додаток А).

Мобільне тестування може бути як ручним, так і автоматизованим, і вибір тестування мобільного додатка залежить від складності програми, бюджету, часових обмежень, навичок команди тестування та наявності інструментів тестування. Поєднання як ручного, так і автоматичного тестування може забезпечити найефективнішу стратегію тестування.

Проте нині 21 столітті. Завдання ручного тестувальника полягає у виконанні регресійних тестів, які дуже повторюються. І очевидно, що робити це вручну набагато повільніше та схильне до помилок порівняно з тим, що потенційно може зробити комп’ютер. Отже, немає причин, чому будь-яке повторюване завдання не може бути повністю автоматизовано, особливо у високотехнологічному середовищі. І саме тому в даній роботі створимо систему гарантування якості мобільного додатка та обґрунтуємо застосування автоматизованого тестування.

РОЗДІЛ 1. АСПЕКТИ СУЧАСНИХ ПІДХОДІВ ТА МЕТОДІВ ТЕСТУВАННЯ

1.1. Загальне поняття про тестування програмного забезпечення

Щоб зрозуміти, що відбувається, нам потрібно почати з самого початку, з людей, які проєктують системи. Чи роблять вони помилки? Звичайно, що так. Люди роблять помилки, тому що вони схильні до помилок, але існує також багато умов, які роблять помилки більш імовірними. Такі умови, як терміни, складність систем і організацій, а також мінлива технологія – все це тисне на розробників систем і збільшує ймовірність помилок у специфікаціях, дизайні та програмному коді. З цих помилок зазвичай починаються серйозні системні збої. Якщо документ із помилкою використовується для визначення компонента, компонент буде несправним і, ймовірно, демонструватиме неправильну поведінку. Якщо цей несправний компонент вбудовано в систему, система може вийти з ладу. Хоча несправність не завжди гарантована, ймовірно, що помилки в специфікаціях призведуть до несправних компонентів, а несправні компоненти спричинять збій системи.

Помилка (error) призводить до дефекту, який може спричинити спостережуваний збій [21].

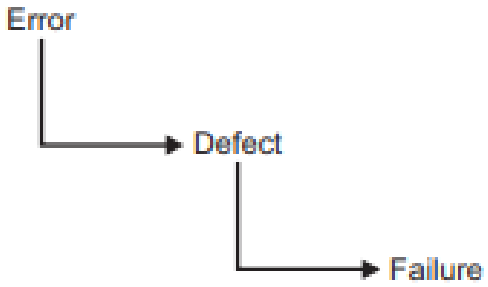


Рис.1.1. Ефект ерори

Якщо ми хочемо уникнути невдачі, ми повинні або уникати помилок і несправностей, або знаходити їх і виправляти. Тестування може сприяти як

уникненню, так і виправленню, як ми побачимо, коли розглянемо процес тестування трохи детальніше.

У бізнесі, техніці та виробництві якість має прагматичне тлумачення як перевага чогось; вона також визначається як придатність для мети [3]. Якість є умовним і певною мірою суб'єктивним атрибутом, який різні люди можуть розуміти по-різному:

- Відповідність вимогам.
- Відсутність відповідності означає відсутність якості.
- Наявність того, що задовольняє клієнтів і користувачів, і відсутність того, що не задовольняє.

На рівні компанії якість програмного забезпечення підтримується через впровадження низки процесів, методів та практик. Є декілька ключових аспектів такі, як стандарти та методології, тестування та QA, контроль версій та управління конфігураціями, перегляд коду (Code Review) і навчання та професійний розвиток. Також важливу роль відіграють метрики та ключові показники продуктивності (KPI) зі сторони проджект менеджменту для контролю якості програмного забезпечення. Ось кілька прикладів метрик та KPI, які можна використовувати:

- Метрики покриття тестами: Наприклад, відсоток покриття коду тестами або кількість успішно пройдених тестів.
- Метрики дефектів: Ці метрики оцінюють ефективність процесу виявлення та усунення дефектів. Вони можуть включати кількість виявлених дефектів, середній час усунення дефектів або час простою через дефекти.
- Метрики продуктивності: Наприклад, середній час відгуку системи або кількість запитів, оброблених за одиницю часу.
- Метрики виконання графіка проекту: Наприклад, відсоток завершення робіт, тривалість завдань або кількість затримок.

- Ключові показники продуктивності (KPI): Наприклад, кількість задоволених клієнтів, рівень виконання вимог, вартість якості або кількість повторних дефектів.

Однак у кожній організації є відділ забезпечення якості для виконання різноманітних заходів системи якості. Забезпечення якості також відоме як тестування якості. Попри це тестування програмного забезпечення та QA (Quality Assurance) – це не одне й те саме. Вони мають різні цілі, особливості та способи застосування.



Рис.1.2. Відміності між поняттями QA, QC, testing

Забезпечення якості зосереджено на вдосконаленні процесу розробки програмного забезпечення та забезпеченні його ефективності та ефективності відповідно до стандартів якості, визначених для програмних продуктів. Тестування програмного забезпечення – це спосіб дослідження системи, щоб перевірити, як вона працює, і знайти можливі дефекти. Тестування дає клієнтам можливість побачити, чи відповідає розроблений продукт їхнім очікуванням щодо дизайну, сумісності, функціонування тощо і має цінність тоді, коли воно приносить цінність іншій частині життєвого циклу розробки програмного забезпечення. Але разом з тим забезпечення якості та тестування працюють в одному напрямку та зосереджені на якості продукції. Професійне ставлення до спільної мети разом із міцною комунікацією забезпечить реалізацію поставленої мети [21].

Кожна модель життєвого циклу розробки програмного забезпечення має значний вплив на те, як проводиться тестування та як аналітики тестування беруть участь у процесі розробки програмного забезпечення. Наприклад, у життєвих циклах Agile тестувальник має бути залучений від початку проєкту до часу планування випуску. Тобто QA проходить весь життєвий цикл продукту (SDLC).

Життєвий цикл розробки програмного забезпечення (Життєвий цикл IT-проєкту) стосується проєктування, розробки та тестування програмного забезпечення високої якості. SDLC (точніше, життєвий цикл розробки програмного забезпечення) визначає повний цикл розробки. Іншими словами, SDLC поєднує різні операції, які є частиною планування, створення, тестування та розгортання будь-якого програмного продукту.

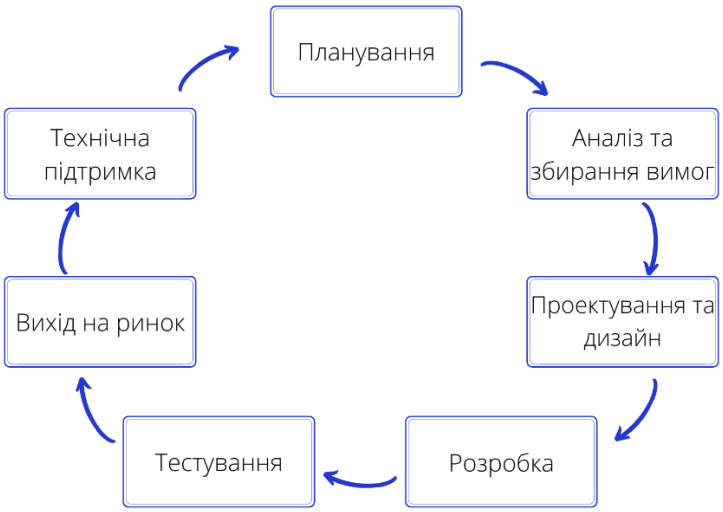


Рис. 1.3. Життєвий цикл програмного забезпечення

Всупереч усім міркуванням, тестування програмного забезпечення – це не просто окрема діяльність чи операція. Воно складається з серії заходів, які проводяться методологічно, щоб допомогти сертифікувати програмний продукт, і підпорядковується процесу забезпечення якості. Зазначимо, що процес забезпечення якості описується життєвим циклом тестування програмного забезпечення (STLC). SDLC і STLC певною мірою взаємопов'язані, або можна сказати, що один є попередником іншого.

Життєвий цикл тестування програмного забезпечення (STLC) – це послідовність конкретних дій, які виконуються під час процесу тестування, щоб забезпечити досягнення цілей якості програмного забезпечення. STLC передбачає як верифікацію, так і валідацію. Основною метою STLC є виявлення та документування будь-яких дефектів або проблем у програмному додатку якомога раніше в процесі розробки. Це дозволяє розглядати та розв’язувати проблеми до того, як програмне забезпечення буде випущено для громадськості.

Етапи STLC включають планування тесту, аналіз тесту, дизайн тесту, налаштування тестового середовища, виконання тесту, закриття тесту та повторне тестування дефектів. Кожен із цих етапів включає певні дії та результати, які допомагають забезпечити ретельне тестування програмного забезпечення та відповідність вимогам кінцевих користувачів.



Рис. 1.4. Життєвий цикл тестування

Загалом STLC є важливим процесом, який допомагає забезпечити якість програмного забезпечення та забезпечує систематичний підхід до тестування. Це дозволяє організаціям випускати високоякісне програмне забезпечення, яке відповідає потребам їхніх клієнтів, що в кінцевому підсумку призводить до задоволеності клієнтів і успіху бізнесу.

Характеристиками STLC є:

- STLC є фундаментальною частиною життєвого циклу розробки програмного забезпечення (SDLC), але STLC складається лише з фаз тестування.
- STLC починається, щойно вимоги визначені або документ із вимогами до програмного забезпечення надано зацікавленим сторонам.
- STLC дає покроковий процес для забезпечення якості програмного забезпечення.
- На початкових етапах STLC, поки розробляється програмний продукт або додаток, команда тестування аналізує та визначає область тестування, критерії входу та виходу, а також тестові випадки. Це допомагає скоротити час випробувального циклу, а також покращує якість продукції. Щойно фаза розробки завершиться, команда тестування готова з тестами та починає виконання. Це допомагає знаходити помилки на ранній стадії.

1.2. Вхідні та вихідні дані

Процес тестування програмного забезпечення, якщо він не зв’язаний жодними умовами чи критеріями, його можна розпочати в будь-який час і проводити нескінченно або дуже коротко. Це може призвести до негативних наслідків, таких як відсутність цілей і завдань, нездатність дотримуватись термінів, перевищення бюджету, неефективне тестування через неадекватні вимоги та розуміння продукту.

Команда з контролю якості повинна знати та розуміти, коли доцільно проводити процес тестування та в який момент процес потрібно завершити. Треба знати, з чого почати і де зупинитися. Коротко кажучи, слід визначити та спланувати процес тестування разом із критеріями входу та виходу з нього.

Критерії входу – це умова або набір умов, які повинні існувати або бути виконані, щоб почати процес. Щоб розпочати діяльність з тестування програмного забезпечення, на етапі планування визначаються та конкретизуються деякі умови

та фактори, які необхідно виконати, щоб позначити початок діяльності з тестування програмного забезпечення.

Розгляньмо деякі умови або ситуації, які можна розглядати як критерії входу для початку тестування (Табл.1.1).

Критерій виходу визначає виконання або припинення тестового завдання. Критерії виходу – це умова або набір умов, які передбачають завершення діяльності або досягнення цілей і мети. Подібно до критеріїв входу, критерії виходу також визначаються та окреслюються на етапі планування тесту. Однак критерії виходу важче визначити порівняно з критеріями входу. Нижче наведено деякі з можливих критеріїв виходу, які можна використовувати для позначення завершення тестування (Табл.1.1).

Таблиця 1.1.

Приклади критеріїв входу та виходу

Критерії входу	**Критерії виходу**
Наявність тестового середовища для виконання тесту. Наявність належних тестових даних. Наявність належних перевірених даних. Тестувальники навчені та необхідні ресурси доступні. Вимоги чітко визначені та затверджені. Тестовий дизайн і план документації готові.	Усі тестові випадки виконано. Було досягнуто достатнього покриття вимог і функціональних можливостей. Усі помилки високого рівня серйозності та пріоритету було виправлено. Визначена територія високого ризику була взята та перевірена. Бюджет було вичерпано. Дедлайни були досягнуті.

В ідеальному світі неможливо перейти до наступного етапу, доки не буде виконано критерії виходу для попереднього етапу. Але і практично це не завжди можливо.

1.3. Мета тестування

Тестування програмного забезпечення має різні цілі та завдання. Основні цілі тестування програмного забезпечення такі:

- Пошук дефектів, які можуть бути створені програмістом під час розробки програмного забезпечення.
- Завоювання довіри та надання інформації про рівень якості.
- Щоб запобігти дефектам.
- Щоб переконатися, що кінцевий результат відповідає вимогам бізнесу та користувачів.
- Щоб переконатися, що він задовольняє BRS, тобто специфікацію бізнес-вимог, і SRS, яка є специфікацією системних вимог.
- Завоювати довіру клієнтів, надаючи їм якісний продукт.

Тестування програмного забезпечення допомагає завершити програмне забезпечення або продукт відповідно до вимог бізнесу та користувачів. Дуже важливо мати хороше тестове покриття, щоб повністю перевірити програмне забезпечення та переконатися, що воно працює добре та відповідає специфікаціям.

При визначенні тестового покриття тестові випадки повинні бути добре розроблені з максимальною можливістю пошуку помилок або дефектів (багів). Тестові випадки мають бути дуже ефективними. Цю мету можна виміряти кількістю дефектів, про які повідомляється в тестових випадках. Чим більше повідомлень про дефекти, тим ефективнішими є тестові випадки.

Процес тестування програмного забезпечення забезпечує належний рівень готовності системи до використання. Щоб забезпечити хороше охоплення, тестування повинно включати різні види тестів, такі як функціональне тестування для перевірки роботи програми, тестування сумісності з різними операційними системами, апаратним забезпеченням та браузерами, тестування продуктивності для перевірки швидкодії програми та тестування навантаження, щоб переконатися в надійності системи та відсутності проблем з блокуванням.

1.4. Рівні та види тестування

Тестування програмного забезпечення здійснюється протягом усього життєвого циклу розробки продукту. Ця процедура допомагає переконатися, що продукт зможе успішно функціонувати на ринку.

Перед релізом будь-який програмний продукт проходить 4 основних рівні тестування, які дозволяють команді розробників переконатися в коректній роботі системи:

- модульне тестування;
- інтеграційне тестування;
- тестування системи;
- приймальні випробування.

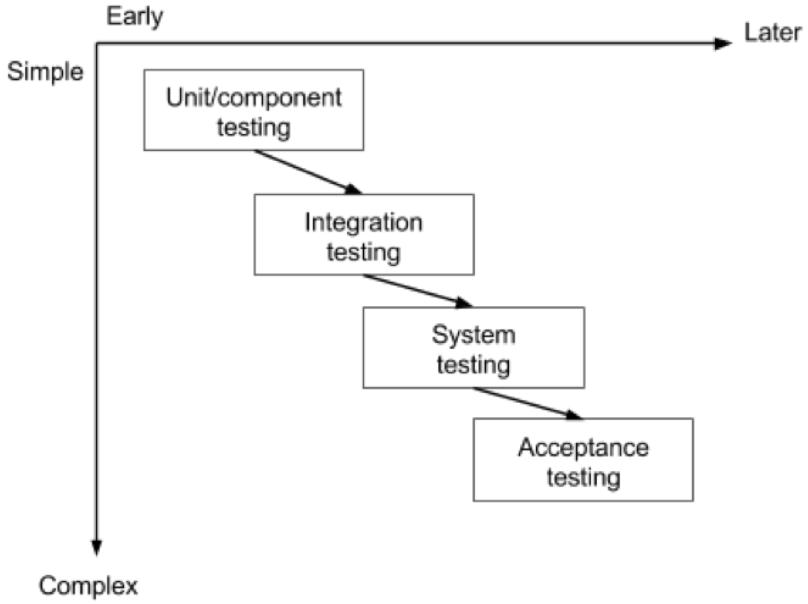


Рис. 1.5. Рівні тестування

Мета всіх цих рівнів тестування – зробити процес тестування програмного забезпечення більш систематичним і легко визначити всі можливі помилки програми [7].

Модульне тестування – це перший етап тестування програмного забезпечення, на якому розробники перевіряють окремі блоки системи, щоб переконатися в їхній придатності до використання.

На наступному рівні, інтеграційному тестуванні, інженери з якості комбінують ці блоки та тестують їх разом у групі, з метою виявлення помилок у взаємодії між системними блоками. Є 4 основні підходи, які використовуються для проведення інтеграційного тестування для клієнтів. Вони включають:

- Тестування інтеграції «зверху вниз».
- Інтеграційне тестування «знизу вгору».
- Інтеграційне тестування великого вибуху.
- Змішане інтеграційне тестування.

Тестування системи охоплює перевірку технічних характеристик та бізнес-логіки програмного забезпечення. Функціональні тести використовуються для перевірки функцій системи, а нефункціональні тести – для оцінки їх ефективності.

Приймальне тестування є останнім етапом перед запуском системи в експлуатацію. Цей тип тестування проводиться тестувальниками з метою допомогти клієнту оцінити продукт і переконатися, що система відповідає вимогам.

Існує два види приймального тестування:

- Альфа-тестування виконується інженерами з контролю якості або розробниками, щоб вивчити розроблене програмне забезпечення з точки зору клієнта.
- Після завершення альфа-тестування ви можете почати бета-тестування, щоб покращити якість продукту. Основна відмінність між бета-тестуванням і альфа-тестуванням полягає в тому, що бета-тестування проводяться групою реальних користувачів.

Підсумовуючи, всі рівні тестування програмного забезпечення є важливими та мають бути завершені до запуску програми. Але тестуючи програми QA інженери звертаються не тільки до рівнів тестування ПЗ (Таблиця Б.1, Додаток Б).

Розповідаючи про рівні тестування, частково розглянемо тему типів, а саме коли йшлось про системний рівень. Згідно з визначенням ISTQB, типи тестування – це «засоби чіткого визначення мети певного рівня для програми або проєкту». Тестувальник зосереджується на конкретній меті тесту під час його виконання. Залежно від цілей розрізняють чотири види тестування ПЗ:

- Функціональне тестування.
- Нефункціональне тестування.
- Базоване на структурі.
- Тестування, пов’язане зі змінами.

Функціональне тестування перевіряє, чи кожна функція програми працює відповідно до специфікації вимог. Функціональне тестування показує «Що робить система». Метою цього тестування є перевірка функціональності системи [7].

Під час функціонального тестування необхідно пам’ятати про п’ять кроків:

- Підготовка тестових даних на основі специфікацій функцій.
- Бізнес-вимоги є вхідними даними для функціонального тестування.
- Знайти вихід функцій на основі функціональних специфікацій.
- Виконання тестових випадків.
- Спостереження за фактичними та очікуваними результатами.

Типи нефункціонального тестування стосуються нефункціональних вимог. Нефункціональне тестування допомагає оцінити готовність системи за різними критеріями, які не охоплюються функціональним тестуванням. На відміну від функціонального тестування, воно показує, «наскільки добре працює система»:

- Тестування інтерфейсу користувача (UI).
- Тестування користувацького досвіду (UX).
- Тестування пам’яті.
- Тестування безпеки.
- Тестування вразливості.

- Кросбраузерне тестування.
- Тестування локалізації.
- Тестування відновлення.

Структурне тестування визначає правильність реалізації програми або коду шляхом тестування структури програмної системи або її компонентів. Під час структурного тестування тестер зосереджується на роботі програмного забезпечення, і цей метод може бути використаний на будь-якому рівні тестування.

Методами структурного тестування є:

- Statement Coverage перевіряє, чи кожен оператор у програмі виконується принаймні один раз під час тестування програми.
- Decision Coverage спрямоване на задоволення критеріїв покриття для кожного логічного шляху через програму.
- Branch Coverage перевіряє, чи кожна умова розгалуження для програми має значення true або false.
- Покриття умов подібне до покриття філії. Основною відмінністю є тести Condition Coverage для умовних і неумовних гілок.

Тестування, пов’язане зі змінами, використовується для перевірки виправлення раніше виявлених помилок та виявлення нових помилок, які можуть з’явитися в нових версіях програмного забезпечення. Існують два підтипи тестування, пов’язаного зі змінами: тестування підтвердження (повторне тестування) та тестування регресії.

Тепер можна сформувати чіткіше уявлення про безліч типів тестування. Деякі з них використовуються щодня величезною кількістю фахівців, інші використовуються дуже рідко. Хоча є сумнів, що типи тестування можна вважати робочими без методів тестування. Оскільки обидва тісно пов’язані між собою. Методи тестування програмного забезпечення описують різні техніки, які використовуються для перевірки працездатності програмної системи [7]. Тестування може бути проведено в різних режимах: в чорному ящику, білому

ящику та сірому ящику. Кожен з цих методів тестування може бути застосований на різних рівнях деталізації, починаючи з високорівневого огляду функціональності системи та закінчуючи детальним аналізом її внутрішньої роботи.

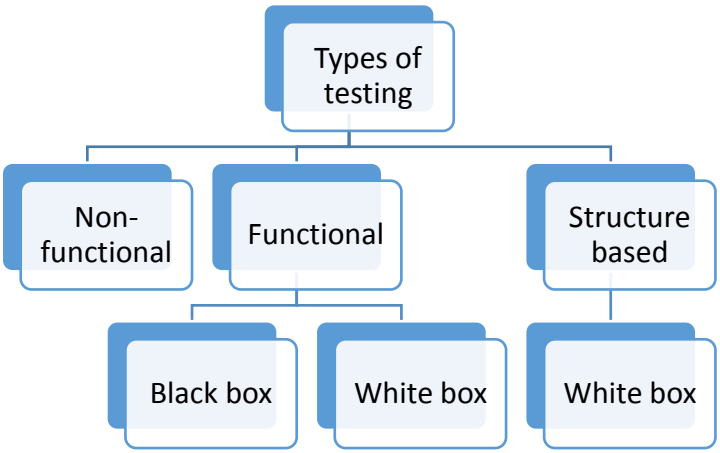


Рис. 1.6. Види та методи тестування

Короткий огляд кожного з цих методів тестування:

1) Чорна скринька – це метод тестування програмного забезпечення, при якому тестер перевіряє функціональність системи без знання її внутрішньої реалізації, зосереджуючись на входах та очікуваних виходах.

2) Біла скринька – це метод тестування програмного забезпечення, при якому тестер має повне знання про внутрішню реалізацію системи та перевіряє її роботу з внутрішньої сторони.

3) Сіра скринька – це метод тестування програмного забезпечення, що поєднує в собі підходи чорної та білої скриньок. Тестер має часткове знання про внутрішню реалізацію системи та використовує цей метод для тестування системної інтеграції.

З іншого боку, типи тестування стосуються різних типів тестування, які можна виконати в рамках цих методів тестування. Наприклад, функціональне тестування, регресійне тестування, тестування продуктивності, тестування безпеки, тестування зручності використання тощо – це різні типи тестування, які можна

виконувати за допомогою методів тестування чорного ящика, білого ящика або сірого ящика.

1.5. Ручне та автоматизоване тестування

Тестування – це величезний домен, який складається з категорій, які вже згадувались раніше: black-box testing, white box testing, system testing, system integration testing, performance testing, and load testing. Деякі з цих категорій тестування краще працюють з автоматизацією, а інші отримують бажані результати за допомогою ручного тестування [14].

Ручне тестування – це процес тестування програмного забезпечення вручну. Коли ми використовуємо людські ресурси для проведення тестування, то ми можемо вважати тестування ручним.

Використання інструментів автоматизації для виконання тестових випадків відоме як тестування автоматизації. Це тип тестування, для якого нам потрібні ресурси зі знаннями сценаріїв тощо.

Ручне тестування має деякі обмеження. Багато разів нам доводиться робити повторювані тестування, і робити речі повторювально може бути нудним. Тестування за допомогою дійсних та недійсних входів може зробити вас розлюченими. І коли мова йде про якість, не можливо йти на компроміс.

З одного боку, ручне тестування допомагає легко визначити хвилинні проблеми, тоді як тестування на автоматизацію потребує повного тестового сценарію та виконання. З іншого боку, автоматизоване тестування допомагає легко писати та запустити тестові випадки для зайвих завдань (Табл.Б.2, Додаток Б).

Тому у більшості випадків оптимальним є поєднання ручного тестування та автоматизації. Жодна компанія не може обійтися без будь-якого виду тестування, і як ручне, так і автоматизоване тестування мають свої переваги та недоліки. Ручне тестування надзвичайно корисне для складних тестових випадків, тоді як автоматизоване тестування є ефективним для простих та повторюваних тестів.

Мета автоматизації полягає у тому, щоб допомогти тестувальникам виконувати тестування швидше та надійніше [22]. Проте, є типи тестів, де автоматизація не допоможе, тому важливо вибирати між ручним та автоматизованим тестуванням залежно від потреб. Інструменти автоматизації розробляються вручну, а їхні можливості обмежені людським фактором. Автоматизоване тестування може здійснювати тільки ті завдання, які були задані, але це не завжди погано, оскільки це гарантує стабільність і надійність. З часом інструменти автоматизації розвиватимуться та стануть більш розумними, але все ж таки ручне тестування є невіддільною частиною процесу тестування.

1.6. Тестування мобільних додатків

Мобільні програми є прикладним програмним забезпеченням, спеціально розробленим для роботи на мобільних пристроях, таких як смартфони, планшети та інші мобільні пристрої. Вони мають робочий графічний інтерфейс користувача (GUI) і виконують певні завдання, що забезпечують користувачам функціональність та зручність використання на мобільних пристроях.

Завдяки широкому розповсюдженню смартфонів та планшетів, на ринку доступно величезна кількість мобільних додатків. Лише в одних магазинах додатків Google Play Store та Apple App Store налічується майже 4 мільйони додатків, що створює велику конкуренцію та розмаїття варіантів для клієнтів.

Оскільки швидкість розробки та випуску мобільних додатків є важливою для зайняття свого місця на ринку, спеціалізовані технології тестування мобільних додатків використовуються для прискорення процесу розробки та забезпечення якості.

Застосування спеціалізованих технологій тестування мобільних додатків допомагає розробникам прискорити життєвий цикл розробки, виявляти та виправляти проблеми, забезпечувати високу якість та забезпечувати швидкий вихід на ринок.

1.6.1. Типи мобільних додатків

Оскільки програмна індустрія постійно розвивається, компанії повинні слідкувати за технологічними тенденціями, щоб використовувати нові додатки та платформи для поліпшення клієнтського досвіду. Останній звіт Gartner прогнозує, що до 80% організацій, які надають послуги клієнтам, відмовляться від розробки власних мобільних додатків на користь платформ обміну повідомленнями, щоб забезпечити більш зручний та різноманітний досвід для клієнтів. На ринку існує безліч типів програм, але ми зосередимося на розумінні нативних, гібридних та вебдодатків.

- Нативні програми спеціально розроблені для конкретної мобільної платформи, зазвичай для Android або iOS, і встановлюються через магазин додатків, такий як Google Play Store або Apple App Store. Ці програми можуть бути чистими нативними додатками, розробленими на Java/Kotlin для Android або Objective-C/Swift для iOS, або кросплатформними нативними додатками, створеними з використанням фреймворків, таких як Flutter, ReactNative та NativeScript.
- Адаптивні вебпрограми призначені для доступу в мобільному браузері. Вебпрограми можуть бути адаптивною версією вебсайту або прогресивною вебпрограмою (PWA), яка додає додаткові функції, зручні для мобільних пристроїв.
- Гібридні програми є посереднім варіантом між нативними та вебпрограмами. Вони можуть бути встановлені через магазин додатків, так само як нативні програми, і мають власні функції, але, як правило, вони використовують вебтехнології, обгортаючи їх у рідну оболонку. Гібридні програми розробляються з використанням веб-технологій, таких як HTML, CSS та JavaScript, і використовують фреймворки, наприклад, Apache Cordova (PhoneGap) або Ionic.

Таблиця 1.2.

Порівняння різних типів мобільних додатків

Тип додатка	Переваги	Недоліки
Власні/рідні (Native)	Висока продуктивність, можливість взаємодії зі всіма функціями пристрою, можливість доступу до спеціалізованих API та SDK для розширення можливостей, можливість працювати в автономному режимі, висока якість відображення інтерфейсу	Високі витрати на розробку і підтримку, вимоги до платформи, складність процесу розробки і тестування
Адаптивні вебпрограми (Web)	Швидкий процес розробки, масштабованість та доступність для користувачів, можливість оновлювати додаток без необхідності встановлення оновлення, можливість працювати на різних платформах	Обмежені можливості доступу до пристрою та його функцій, менша продуктивність, менша якість відображення інтерфейсу, залежність від інтернет-з'єднання
Гібридні (Hybrid)	Комбінація переваг нативних і вебдодатків, можливість використовувати вебтехнології для розробки та доступу до функцій пристрою, можливість працювати в автономному режимі, швидший процес розробки, масштабованість та доступність для користувачів	Менша продуктивність, обмежені можливості доступу до деяких функцій пристрою, менша якість відображення інтерфейсу, вимоги до платформи, обмежені можливості налаштування і розширення.

Важливо пам'ятати, що вибір типу додатка залежить від специфіки проєкту і його вимог, а також від доступних ресурсів та технічних інструментів.

1.6.2. Прилади для тестування мобільних додатків

Організації, які прагнуть забезпечити якість мобільних додатків, стикаються з важливим вибором щодо того, де проводити тестування мобільних пристроїв. Команди розробників на великих підприємствах за замовчуванням використовують реальні мобільні пристрої. Хоча це дає точніші результати тестування, воно не є ідеальним для масштабування та автоматизації тестування.

Намагаючись відійти від тестування на фізичних пристроях, деякі організації перейшли на використання емуляторів і симуляторів для тестування мобільних додатків. Однак найкраще об’єднати реальні пристрої та віртуальні пристрої (мобільні емулятори та симулятори) для найповнішої стратегії тестування мобільних додатків.

Таблиця 1.3.

Порівняння справжнього та віртуальних пристроїв

	Справжній пристрій	**Емулятори/Симулятори**
Простота надання		+
Легко масштабувати		+
Сприяє автоматизації		+
Виявлення апаратних збоїв	+	
Розширене тестування інтерфейсу користувача	+	
Легко обслуговувати		+

Мобільний емулятор, як випливає з терміну, емулює програмне та апаратне забезпечення пристрою на настільному ПК або як частину хмарної платформи

тестування. Це повна повторна реалізація мобільного програмного забезпечення, написаного мовою асемблера машинного рівня. Одним із таких прикладів є емулятор Android (SDK).

Симулятор надає копію інтерфейсу користувача телефону і не представляє його апаратне забезпечення. Симулятор – це часткова повторна реалізація операційної системи, написана мовою високого рівня. Одним із таких прикладів є симулятор iOS для пристроїв Apple.

Справжній (фізичний) пристрій – це апаратне забезпечення (плюс ОС і вбудовані ресурси підтримки), на якому програмне забезпечення працюватиме у виробництві. Для мобільного програмного забезпечення це мобільний телефон або планшет.

Отже, узагальнюючи, можна сказати, що через величезну кількість пристроїв і операційних систем на ринку просто неможливо перевірити все вручну. Крім того, існує багато областей тестування мобільних додатків, наприклад мережевий трафік або продуктивність мобільного сервера, які потребують автоматизації, щоб отримати значущі результати.

1.6.3. Виклики при тестуванні мобільних додатків

Автоматизоване тестування мобільних додатків своєю чергою має свої виклики з точки зору програмного та апаратного забезпечення. Деякі з викликів включають:

- Версії операційних систем: Мобільні додатки можуть працювати на різних версіях операційних систем, що може впливати на їхню сумісність та функціональність. Тому тестування додатків на різних версіях операційних систем є важливим аспектом.
- Різноманітність пристроїв: Мобільні пристрої мають різні характеристики, що може вплинути на функціональність додатка та його

продуктивність. Це може вплинути на функціональність додатку та його продуктивність. Тому тестування додатка на різних пристроях є важливим етапом.

- Обмеження мережі: Деякі додатки можуть працювати тільки в мережі Інтернет, що може бути проблемою для користувачів, які користуються мобільним зв'язком. Тому тестування додатків у різних умовах мережі є важливим етапом.
- Відсутність зв'язку: Користувачі можуть втратити зв'язок з мережею, що може вплинути на роботу додатка. Тому тестування додатків в офлайн-режимі є важливим етапом.
- Розміщення в магазині додатків: Мобільні додатки повинні відповідати політиці магазину додатків, щоб бути прийнятими в нього. Тому тестування додатків на відповідність політиці магазину додатків є важливим етапом.
- Інтерфейс користувача: Мобільні додатки мають свій унікальний інтерфейс користувача, що може бути викликано різноманітними проблемами.
- Безпека: Мобільні додатки можуть бути підвищеним ризиком для безпеки, оскільки вони містять конфіденційну інформацію та можуть бути підключені до різних мереж. Тестування повинно включати в себе тестування безпеки, щоб переконатися, що додаток не має дірок у забезпеченні безпеки, таких як вразливості SQL-ін’єкції, Cross-site scripting (XSS) та інші.
- Різноманітність тестових сценаріїв: Мобільні додатки можуть містити різноманітні функції та функціональність, яка потребує різних тестових сценаріїв. Також, тестування мобільного додатку повинно включати тестування взаємодії додатку з іншими додатками та сервісами, такими як соціальні мережі та поштові сервіси.

Всі ці виклики при тестуванні мобільних додатків потребують відповідного тестування та використання технологій, що враховують унікальні особливості мобільних платформ.

РОЗДІЛ 2. АРХІТЕКТУРА АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ТА ЗАСОБИ АВТОМАТИЗАЦІЇ

2.1. Архітектура автоматизованого тестування

При виборі стратегії тестування потрібно враховувати багато факторів. Однак, одним з основних правил є максимальне використання автоматизованого тестування, оскільки ручне тестування відбувається повільно і не завжди встигає зі всіма процесами. Саме тому для виконання роботи обрано автоматизований процес тестування мобільного додатка, який працює з нативним додатком. Крім того, для тестування використаємо Andoid SDK, який є безплатним і дозволяє виконати тестування на різних пристроях з різними характеристиками. Нижче в розділі подано більше інформації, яку необхідно знати, перед початком реалізації системи гарантування якості на основі автоматизованого тестування.

Автоматизоване тестування ПЗ дозволяє підвищити швидкість тестування ПЗ без шкоди для його якості. Спеціально розроблений фреймворк автоматизованого тестування може допомогти протестувати всі (навіть найменші) частинки продукту та забезпечити охоплення наскрізними тестами для різних сценаріїв використання (Рис. 2.1).

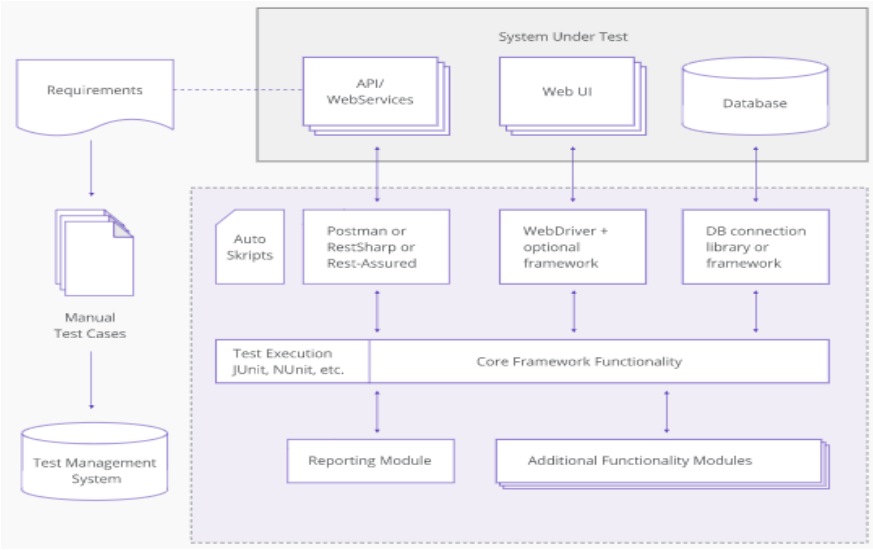


Рис. 2.1. Проста архітектура автоматизації тестування

Автоматизаційний фреймворк або архітектура автоматизації тестування – це набір правил, інструментів та процесів, які допомагають забезпечити комплексну перевірку якості програмного забезпечення на різних етапах його розробки. Метою фреймворків є надання загальної структури, щоб розробники могли повторно використовувати код без необхідності його повторення. Фреймворки дозволяють уникнути непотрібної роботи та значно прискорити процес тестування [18].

Існує кілька різних систем автоматизації тестування, кожна з яких має свою власну логіку.

1) Linear Automation Framework використовується для тестування невеликих програм і має неструктуровані сценарії.

2) Keyword-driven framework розбиває логіку тестування на ключові слова та функції.

3) Modular-driven framework аналізує тестові випадки та перетворює їх на модульні сценарії, які можна повторно використовувати.

4) Behavior-driven framework визначає поведінку додатків за допомогою зрозумілих людині тестових команд та кодифікує їх у тестові випадки за допомогою інструментів BDD.

5) Data-driven framework дозволяє створювати тестові приклади на основі доступних тестових даних, які зберігаються у форматі таблиць. Існує також Hybrid framework, який є комбінацією різних підходів до автоматизації тестування.

6) Hybrid framework – будь-яка комбінація вищевказаних структур AQA.

Загальна архітектура автоматизації тестування (gTAA) була представлена Міжнародною кваліфікаційною радою з тестування програмного забезпечення (ISTQB) і є частиною навчальної програми ISTQB для інженерів автоматизованого тестування (TAE).

Структура автоматизації тестування, відповідно до gTAA, може складатися з рівня визначення тесту, рівня виконання тесту та рівня адаптації тесту. Крім того,

існує рівень генерації тестів, який розглядається як окремий від інфраструктури автоматизації тестування [18].

- Генерація тестів (Test generation): на цьому рівні ми створюємо ефективні тести, що збільшують вірогідність виявлення дефектів та охоплюють більшу кількість сценаріїв тестування. Для цього використовуються відповідні інструменти та методології.
- Визначення тесту (Test definition): на цьому рівні ми визначаємо умови тестування та створюємо тестові випадки на різних рівнях деталізації, з урахуванням пов'язаних даних та процедур. Також можна розробляти тестові набори та реалізації тестів.
- Виконання тесту (Test execution): на цьому рівні виконуємо тестові випадки та набори, вибираємо потрібні тести та використовуємо функції журналювання та звітування.
- Тестова адаптація (Test adaptation): на цьому рівні адаптуємо інтерфейс (API, графічний інтерфейс користувача, служби тощо) до компонента, програми чи системи, що тестується. Для цього використовуються функціональні можливості, необхідні для взаємодії з різними реалізаціями технологій, які підлягають тестуванню.

Усі рівні залежать від трьох застосованих рівнів керування, а саме керування тестуванням, управління проєктом і керування конфігурацією (Рис. 2.2).

Разом вони регулюють відносини між TAS (рішенням для автоматизації тестування) та SUT (системою, що тестується). Або, іншими словами, ці чотири рівні створюють правила та вказівки для автоматизації тестування.

Для конкретного рішення автоматизації тестування (TAS) слід зазначити, що будь-який із цих рівнів може бути присутнім або відсутнім.

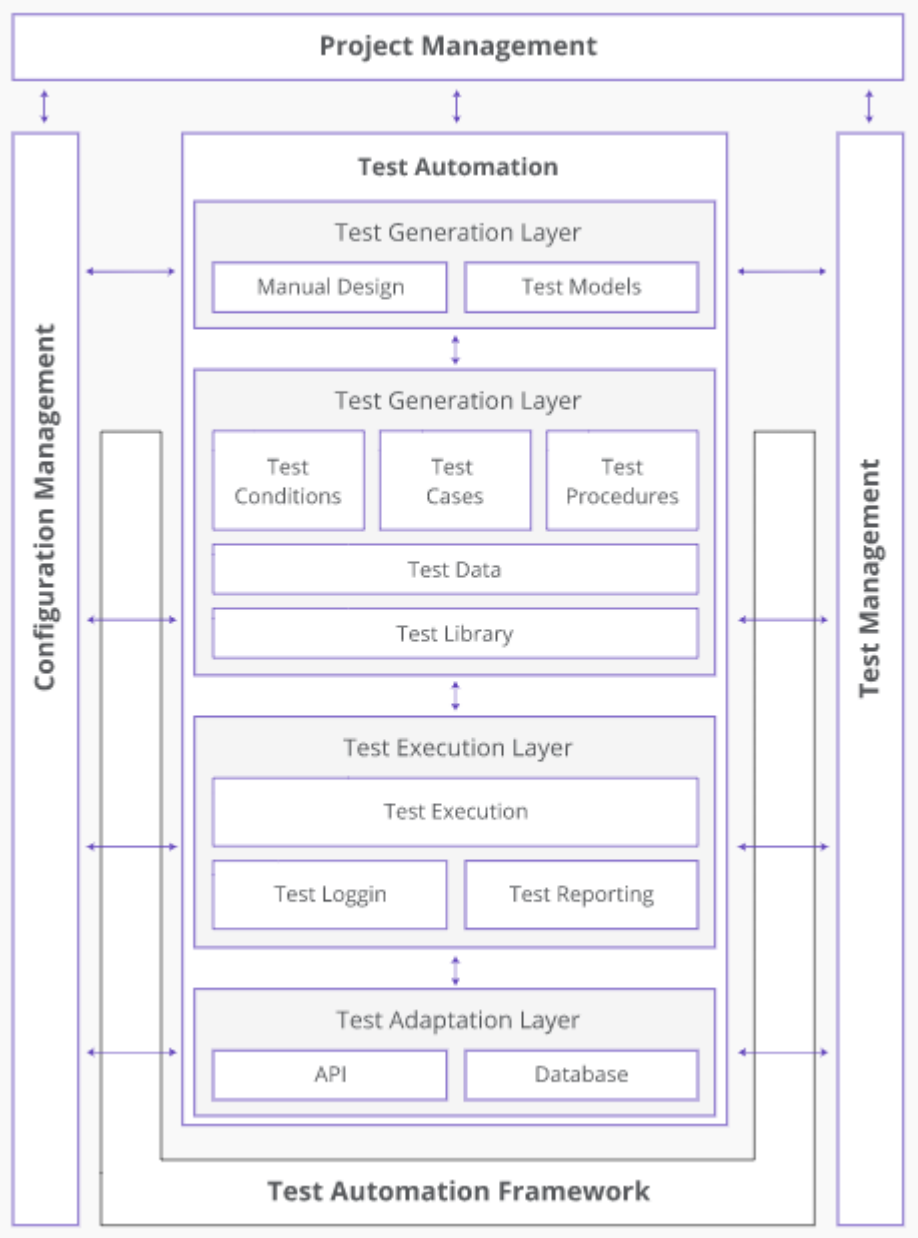


Рис. 2.2. Загальна архітектура автоматизації тестування

Як згадувалося раніше, фреймворк автоматизації тестування – це комбінація процесів і інструментів. Схему процесу уже розглянули. Тепер давайте зосередимося на інструментах автоматизації контролю якості.

Інструментами автоматизованого тестування якості можуть бути:

- З відкритим вихідним кодом – розроблене спільнотою та підтримуване програмне забезпечення, яке можна використовувати, не купуючи жодних ліцензій. Немає обмежень для налаштування.
- Запатентовані інструменти – ліцензоване програмне забезпечення, що пропонує доступ до широкого набору нативних функцій із коробки.
- Індивідуальні рішення – будь-яка комбінація двох вищевказаних варіантів у рамках вашої AQA.

Вибираючи стек автоматизації контролю якості, завжди потрібно враховувати:

- Мови програмування (більшість інструментів контролю якості залежать від мови).
- Операційні системи (використовуються тестувальниками).
- Підтримка платформи (веб, мобільний, планшет тощо).

Узагальнюючи зазначимо, що не обов’язково мати одну архітектуру автоматизації, яка охоплює всі компоненти. Тестування може бути успішним, якщо тестер і засоби автоматизації виконають корисний тест і зроблять правильні висновки на основі результатів. Автоматизація є важливою для того, щоб розширити тестування, виконуючи завдання, які виснажливі або неможливі для людини, або автоматизація яких економічно ефективніша. Різні типи тестів виконуються в різний час, і не всі пов’язані тести потрібно запускати одночасно. Є мало переваг в уніфікованій архітектурі автоматизації, коли існують суттєво різні продукти чи середовища та дуже мало спільного тестового програмного забезпечення для них. Два прості механізми автоматизації часто легше створити та підтримувати, ніж один надто складний [16].

Розібравшись в архітектурі автоматизованого тестування, перейдімо до аналізу стека автоматизації, який обрано для реалізації системи забезпечення якості в даній кваліфікаційній роботі: а саме мови програмування, середовища розробки та інструментів для автоматизації.

2.2. Мова програмування Python

Python – це інтерпретаційна, інтерактивна, об'єктноорієнтована та багатоцільова мова програмування, створена Гідо Ван Россумом наприкінці 1980-х. Це мова програмування – мова, яку люди та комп’ютери можуть зрозуміти. Вона потужна і має простий у користуванні синтаксис, що робить її ідеальною для початківців.

Python – це гнучка мова, яка має можливість писати прості програми, а також створювати великі складні рішення програмування. Часто використовується для розробки бекенду (тобто основи) програми або вебсайту, Python здатний обробляти дані, спілкуватися з базами даних, маршрутизації URL-адрес та забезпечення загальної безпеки [12].

Python також пропонує можливість легко автоматизувати процеси за допомогою сценаріїв, що робить його ключовим для тестування програмного забезпечення, усунення несправностей та відстеження помилок.

До того ж Python відіграє ключову роль у завданнях з наукових даних і використовується для виконання складних статистичних розрахунків, візуалізації даних та створення алгоритмів машинного навчання.

Мова програмування Python – це мова програмування високого рівня. Це означає, що мова розроблена так, щоб її легко писати та читати. Особлива річ у Python полягає в тому, що вона використовує лінійний відступ при кодуванні. Інші мови програмування зазвичай використовують фігурні дужки – такого вигляду {} – але з Python все є в окремих рядках. Це робить мову набагато зрозумілішою [12].

Як вже говорилось, Python є об'єктноорієнтованою мовою програмування, що дозволяє розділяти код на повторно використовувані об'єкти. Це значить, що програміст може заощадити час, не повторюючи код знову та знову. Завдяки компактності Python-коду, програміст може розробляти програму за значно менший час, порівняно з іншими мовами програмування.

Незалежно від того, чи є ви початківцем програмуванням, чи досвідченим програмістом, який розробляє складний додаток, Python – це чудовий вибір через легко зрозумілий синтаксис та величезні можливості.

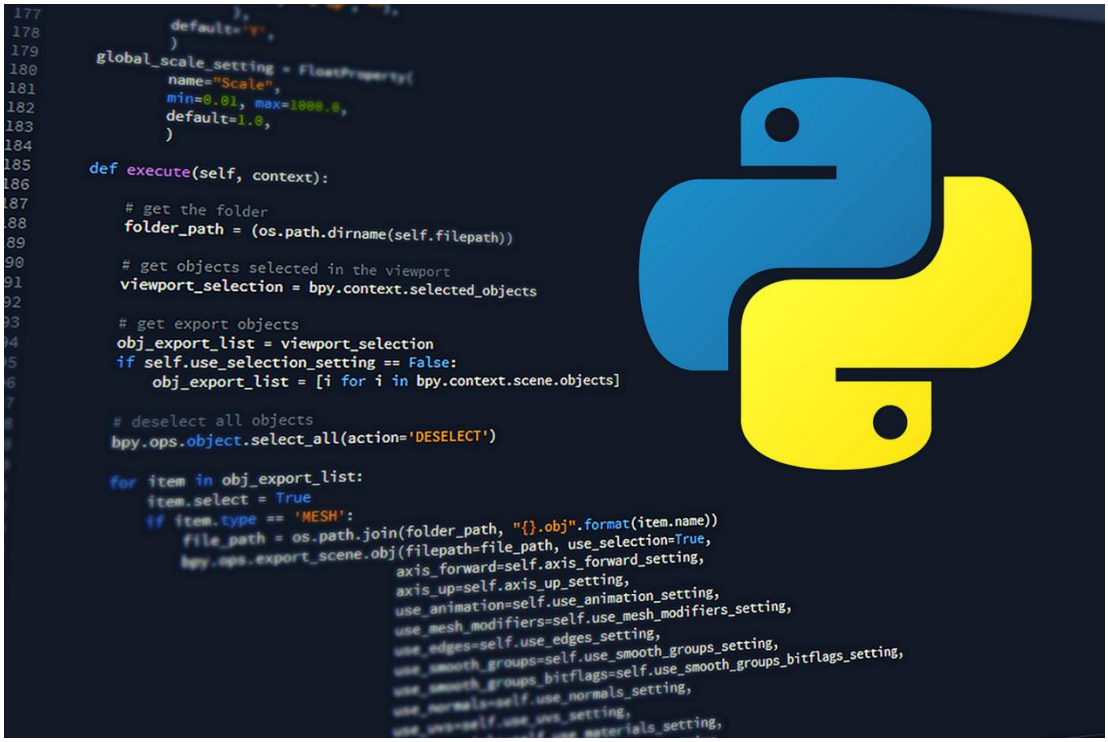


Рис. 2.3. Логотип та частина коду мови програмування Python

Деякі особливості Python, які роблять його непереборним для користувачів:

1) Популярність: Python вважається 4-ою найпопулярнішою та найшвидше висхідною мовою програмування згідно з опитуванням розробників переповнення стека 2019. Python використовується найвідомішими програмними компаніями у світі, такими як Google, YouTube, Instagram, Netflix, Spotify, Quora, Facebook, PayPal, Uber, Reddit та багато іншого.

2) Інтерпретація: Python є інтерпретованою мовою, а це означає, що програми виконуються безпосередньо інтерпретатором, а не компілятором., де вихідний код перетворюється на машинний код перед запуском.

3) OpenSource: Python є вільною мовою з відкритим кодом, що робить її доступною для використання та розповсюдження навіть для комерційних цілей.

4) Портативність: Код Python портативний є портативним, що означає, що він може бути виконаний на будь-якій платформі, що має встановлений інтерпретатор Python.

5) Простота: Стиль кодування Python дуже простий і чистий, що дозволяє легко читати та вивчати. Ця мова програмування використовує менше ключових слів порівняно з іншими мовами програмування, що дозволяє розробникам швидше розуміти і писати програми. Розробники, як правило, використовують її весь час через її акуратну та організовану структуру коду.

Підсумовуючи, можна сказати, що Python – це неймовірно універсальна мова програмування, яка має багато переваг і є популярною серед розробників.

2.3. Інтегроване середовище розробки

Інтегровані середовища розробки (IDE) – це додатки, які полегшують розробку інших додатків. Розроблений для охоплення всіх завдань програмування в одній програмі, однією з головних переваг IDE є те, що вони пропонують центральний інтерфейс з усіма інструментами, які потребує розробник, включаючи:

- Редактор коду: Редактори коду призначені для створення та редагування вихідного коду і відрізняються від текстових редакторів, оскільки спрощують або поліпшують процес написання та редагування коду для розробників.
- Компілятор: компілятори перетворюють вихідний код, написаний мовою читабельною/записаною людиною у формі, яку можуть виконувати комп'ютери.
- Дебагер: дебагери використовуються під час тестування і можуть допомогти розробникам у налагодженні своїх програмних додатків.
- Інструменти автоматизації побудови: Вони можуть допомогти автоматизувати завдання розробника та заощаджити час.

Існує багато різних Integrated Development Environments (IDE), які можна використовувати для програмування на Python, але сама мова має вбудований простий інструмент для початківців – Python IDLE. Цей інструмент доступний для Windows та Mac-користувачів, які встановили Python, а для користувачів Linux, їх можна легко знайти та встановити через диспетчера пакетів. Python IDLE може використовуватися як інтерактивний інтерпретатор або як редактор документів, залежно від налаштувань. Це чудовий варіант для тих, хто тільки починає вивчати Python.

Основні функції IDE Python:

- Здатність дивитися на кілька документів.
- Інтерактивний інтерпретатор із виділенням синтаксису та помилками та звітуванням вводу/виводу.
- Розумні функції відступу та основні текстові редактори.
- Дуже потужний дебагер.

Переваги Python IDE:

- Досить простий.
- Працює без сервера або браузера.
- Має вбудований дебагер.
- Може бути налаштованим відповідно до вподобань користувачів.

Недоліки Python IDE:

- Файли, створені за допомогою простою Python, не доступні з будь-якого пристрою, крім того, на якому вони були створені, якщо вони не будуть скопійовані або надіслані на інший пристрій.
- Зміни не зберігаються автоматично при програмуванні.
- Менш перевершує сучасників.

2.3.1. Pycharm

У сферах промисловості більшість професійних програмістів використовують Pycharm, який вважається найкращим IDE для розробки на Python. Цей інструмент був розроблений компанією JetBrains з Чехії та підтримується на різних платформах. Pycharm надає щоденні поради щодо покращення вашої роботи з цим інструментом, що є дуже корисною функцією. Інструмент поставляється у двох версіях: безплатній версії спільноти та платній професійній версії. Крім того, у Pycharm є багато інших корисних функцій:

- Він вважається інтелектуальним редактором коду, швидким та безпечним рефакторингом та розумним кодом.
- Особливості для налагодження, профілювання, віддаленої розробки, тестування коду, завершення автоматичного коду, швидкого виправлення, виявлення помилок та інструментів бази даних.
- Підтримка популярних вебтехнологій, вебфреймворків, наукових бібліотек та контролю версій.
- Безплатно або оплачується: У Pycharm доступне безкоштовне видання спільноти, але повна, професійна версія є платною. Крім того, існує безкоштовна пробна версія, а також окреме навчальне видання для студентів.
- Системна сумісність: Windows, MacOS, Linux.
- Він також дозволяє інспекції коду та підтримує VCS. Pycharm має одну з найбільш розвинутих функцій інспекції коду серед усіх інтегрованих середовищ розробки для Python, яка допомагає уникнути помилок, які можна запобігти, завдяки статичному аналізу коду та аналізу виконання коду.

2.3.2. Android SDK

Для розробки системи автоматизованого тестування необхідно не лише середовище розробки для написання коду, але й Android SDK, яке інтегрується з Android Studio, інтегрованим середовищем розробки.

Операційна система Android використовується на багатьох різноманітних пристроях з різними розмірами екранів, потужністю обробки, обсягом пам'яті, датчиками та іншими характеристиками. Крім того, існує велика ймовірність того, що користувачі системи матимуть різні версії Android, встановлені на своїх пристроях. На щастя, Android SDK надає нам чудову альтернативу у вигляді емулятора, який може створювати віртуальні пристрої зі своїми зазначеними обмеженнями.

Щоразу, коли Google випускає нову версію, також випускається відповідний SDK. Для роботи з Android розробники повинні завантажувати та встановити SDK кожної версії для конкретного пристрою.

Цей SDK забезпечує вибір інструментів, необхідних для створення додатків для Android, і забезпечує процес, який проходить максимально плавно. Незалежно від того, чи створюється додаток за допомогою Java, Kotlin або C#, щоб запустити його на будь-якому пристрої Android, потрібен SDK.

AVD Manager – це частина SDK Manager для створення та управління створеними віртуальними пристроями. Віртуальний пристрій Android (AVD) – це конфігурація пристрою, яка працює на емуляторі Android.

Android Emulator – це інструмент, який імітує пристрої Android на комп’ютері, що дозволяє розробникам тестувати програми на різних пристроях та рівні API Android, не потребуючи фізичних пристроїв для кожного. Емулятор постачається з конфігураціями для різних телефонів Android, планшетів, носіння ОС та телевізійних пристроїв Android.

Емулятор Android надає майже всі можливості реального пристрою Android. Можна виконати такі заходи:

- імітувати різні швидкості мережі;
- вказувати розташування пристрою;
- моделювати апаратні датчики, такі як обертання;
- доступ до Google Play Store та багато іншого.

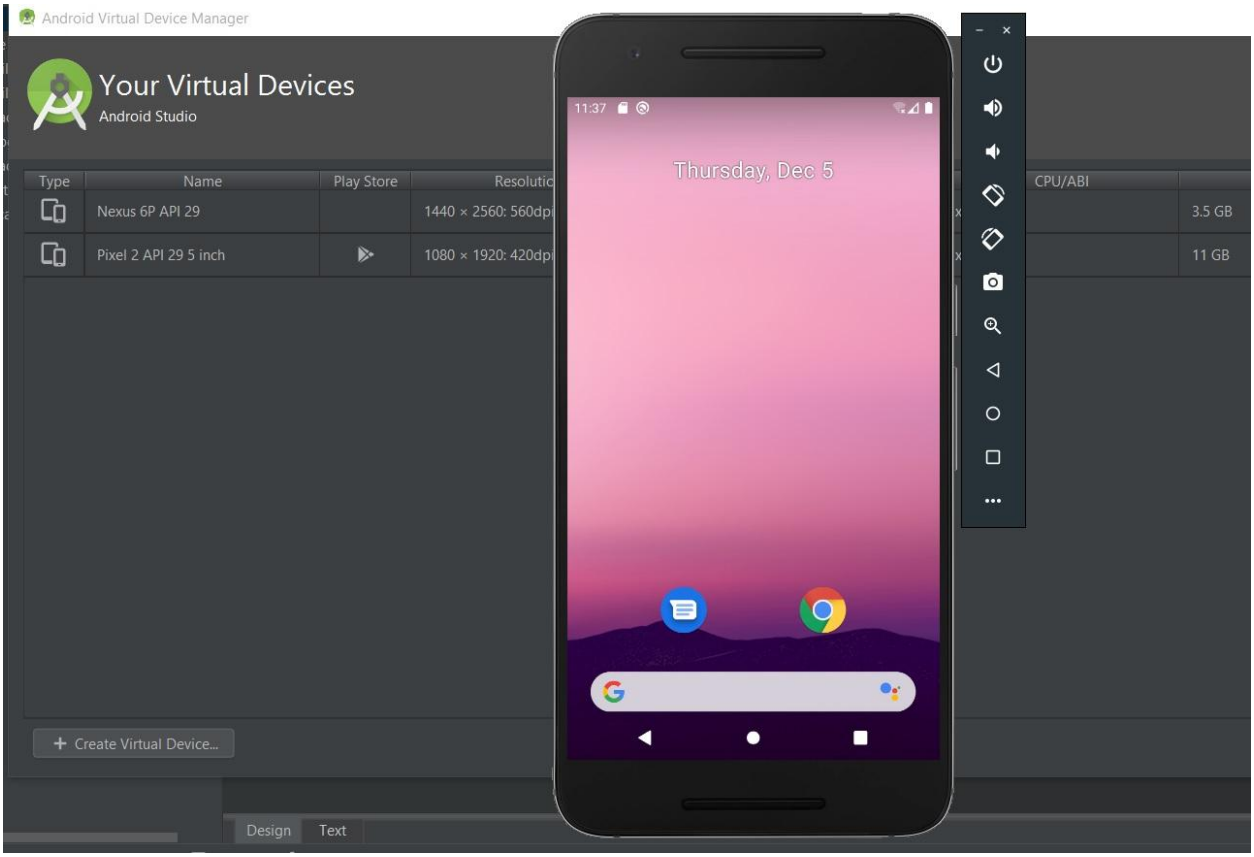


Рис. 2.4. AVD Manager. Емулятор Android

Часто швидше і простіше перевірити свою програму з емулятором, а не використовувати фізичний пристрій.

Та попри все деякі поведінки, як, наприклад, дзвінок, можуть бути належним чином перевірені тільки на фізичному пристрої. Аналогічно, набагато простіше перевірити можливості Bluetooth на фізичних пристроях.

Це означає, що стає абсолютно необхідним мати можливість перевірити свої програми на фізичних пристроях. Фізичні з'єднання пристрою можна здійснити або за допомогою USB, або над Wi-Fi. Можна під'єднати Android Studio до фізичних пристроїв. Підключати пристрої можна за допомогою USB, а також спарювання Wi-Fi. Але також можна використовувати зовнішні програми/додатки для роботи з Android-пристроями через комп'ютер, наприклад Vysor, і допомагає цьому додатку також Android SDK, оскільки для коректної роботи додатка потрібно встановити

USB-драйвери для підключення Android-пристроїв до комп'ютера. Деякі з цих драйверів входять до складу Android SDK. Також Vysor може використовувати ADB (Android Debug Bridge), що також є частиною Android SDK, для забезпечення з'єднання між комп'ютером та Android-пристроєм.

2.4. Фреймворк Pytest

Python пропонує три типи фреймворків:

- Unittest: це тестувальний фреймворк, який побудований в стандартній бібліотеці.
- Nose: він розширює unittest, щоб полегшити тестування.
- Pytest: це фреймворк, який дозволяє легко писати тестові випадки в Python.

Pytest – це фреймворк, який дозволяє легко писати, тестувати та масштабувати масштаб для підтримки складного тестування для додатків та бібліотек. Це найпопулярніший пакет Python для тестування. Основою для багатої екосистеми тестування є плагіни та розширення [9].

Те, як Pytest розроблений, – це дуже розширювана система, прості для запису плагінів, і в Pytest є багато плагінів, які використовуються для різних цілей. Тестування дуже важливе, перш ніж доставити код у виробництві.

Це зрілий повнофункціональний інструмент Python, який допомагає писати кращі програми.

Особливості Pytest:

- Не вимагає використання API.
- Можна використовувати для запуску Unit тестів.
- Надає корисну інформацію про відмови без використання дебагерів.
- Можна записати як функцію або метод.
- Має корисні плагіни.

Переваги Pytest:

- Це відкритий код (OpenSource).
- Він може пропустити тести та автоматично виявляти тести.
- Тести проводяться паралельно.
- Конкретні тести та підмножини тестів можна запустити з програми.
- Почати легко, оскільки він має дуже простий синтаксис.

Перед випуском програмного продукту виробництво багато програмістів здійснюють автоматизоване тестування.

Рекомендується спробувати Pytest, якщо ще не мали змоги використовувати його. Його філософія та функціонал дозволять зробити досвід тестування більш продуктивним та приємним. Завдяки Pytest, загальні тестові завдання вимагають меншого коду, а складніші тести можна легко розширювати за допомогою різноманітних команд та плагінів, що збільшує ефективність виконання тестів. Крім того, Pytest може автоматично запускати наявні тести, включаючи ті, що були написані з використанням Unittest [9].

2.5. Інструмент для автоматизації Selenium

Для тестувальників у всьому світі Selenium є першим вибором для виконання автоматизованих тестів.

Selenium є безплатним інструментом автоматизації тестування з відкритим кодом, який підтримує багато скриптових мов, таких як Python, C#, Java, Perl, Ruby, JavaScript, то залежно від програми, яку необхідно протестувати, можна вибрати відповідну мову. Його API досить легкий у використанні, а також він сумісний з декількома клієнтськими бібліотеками, що дозволяє командам легко проводити тестування між різними браузерами.

Але ми не можемо безпосередньо використовувати Selenium для автоматизованого тестування мобільних додатків. Однак існують фреймворки, які підтримуються Selenium для автоматизації тестування мобільних додатків.

Selenium використовується в поєднанні з сервером Appium для керування тестами автоматизації браузера на мобільних пристроях. На високому рівні:

- Сервер Appium використовує дротовий протокол JSON для прийняття тестових команд у клієнта у вигляді запитів HTTP RESTFUL.
- На основі потрібних можливостей, сервер Appium, виконує ці команди на певних мобільних платформах за допомогою нативних драйверів автоматизації (наприклад, Xcuitest для iOS та UI Automator для Android).
- Appium тягне своє коріння від Selenium і використовує той самий API WebDriver для керування мобільними браузерами на Android та iOS. Таким чином, Appium робить мобільне тестування за допомогою Selenium WebDriver простіше для команд або осіб, які мають попередній досвід роботи з Selenium.

2.6. Інструмент автоматизації Appium

Appium – наступний інструмент автоматизації, який потрібен для розробки системи.

Appium – це платформа автоматизації тестування з відкритим кодом, яка використовується для тестування автоматизації мобільних пристроїв на різних платформах, таких як iOS і Android. Він також підтримує автоматизацію мобільного веббраузера та має широкий вибір мов для автоматизації. Його підтримує Sauce Labs, і його перший офіційний запуск відбувся в травні 2014 року. У 2016 році Sauce Labs пожертвувала Appium JS Foundation, щоб гарантувати, що він завжди залишатиметься відкритим кодом [8].

Кілька років тому Appium був лише інструментом CLI, і його в основному встановлювали за допомогою менеджера пакетів вузлів. Завдяки недавній розробці команди розробників Appium випустив настільну програму на основі графічного інтерфейсу користувача, яку можна інсталювати в різних дистрибутивах Linux, Microsoft Windows і MAC OS.

Варто також відзначити, що Appium є кросплатформною програмою у всіх її сенсах:

1) Appium можна використовувати для тестування мобільних додатків незалежно від того, працюють вони на iOS чи Android.

2) Appium можна використовувати для тестування гібридних, нативних і вебдодатків.

3) Appium працює на Windows, Linux і MAC.

Серцем Appium є сервер, написаний на node.js. Сервер працює за архітектурою клієнт-сервер. У нього входять 3 компоненти:

- Клієнт Appium.
- Сервер Appium.
- Кінцевий пристрій.

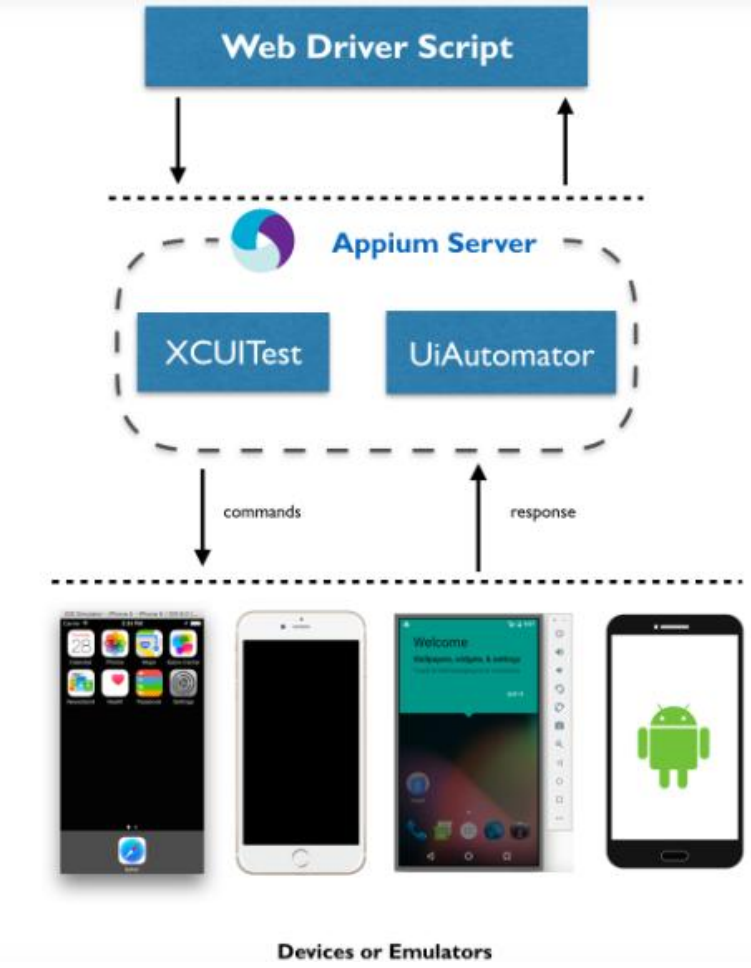


Рис. 2.4. Клієнто-серверна архітектура Appium

За архітектурою клієнт-сервер, програма Appium на стороні клієнта містить налаштування та сценарій тестування, які відправляються на сервер у вигляді JSON-команд. Сценарій автоматизації перетворюється у формат JSON за допомогою вбудованих файлів jar на клієнті.

Сервер Appium отримує команду та встановлює з'єднання з необхідним пристроєм. Після встановлення з'єднання сервер запускає виконання тестів на кінцевому пристрої. Кінцевий пристрій відповідає на запит у формі HTTP до Appium.

Коли тестові випадки виконуються на мобільному пристрої/емуляторі, він заповнює журнал усіх дій (the log of all the actions), які виконуються на пристрої/емуляторі.

Appium працює по-різному на iOS та Android. Оскільки цей інструмент є міжплатформовим, необхідно було створити механізм для розрізнення запитів на сеанси на обох операційних системах. Цю конкретну проблему було вирішено шляхом використання об'єктів JSON, які називаються "Бажаними можливостями" (Desired Capabilities), які є парою ключ-значення і відрізняють встановлення сеансу для тестування програми Android від програми iOS. Сервер може легко розрізнити дві операційні системи за допомогою таких аргументів, як:

- platformName: вказує на назву мобільної операційної системи, на якій буде виконуватись тестування, тобто "Android" для Android-пристроїв. Цей параметр дуже важливий для коректної настройки тестового середовища, оскільки різні операційні системи можуть вимагати різних налаштувань та методів взаємодії з додатком;
- deviceName: вказує на назву тестового пристрою, з яким буде взаємодіяти Appium;
- appPackage: вказує на ідентифікатор пакету мобільного додатку, який тестується. Цей параметр необхідний для того, щоб Appium міг знайти та запустити

відповідний додаток на пристрої. Вказується у вигляді рядка, який містить ім'я пакету додатка, наприклад "com.example.myapp";

- appActivity: ця властивість вказує на активність мобільного додатку, який повинен бути запущений під час тестування;
- noReset: вказує на те, чи потрібно виконувати повне очищення даних мобільного додатку перед кожним запуском тестів. Якщо цей параметр встановлено в true, то Appium не буде очищати дані додатку після кожного запуску тестів, що може зекономити час у випадку, коли додаток має довгий час завантаження або ініціалізації. Якщо ж параметр noReset встановлено в false або не вказано, Appium буде повністю очищати дані мобільного додатку перед кожним запуском тестів, щоб забезпечити консистентність результатів тестування.

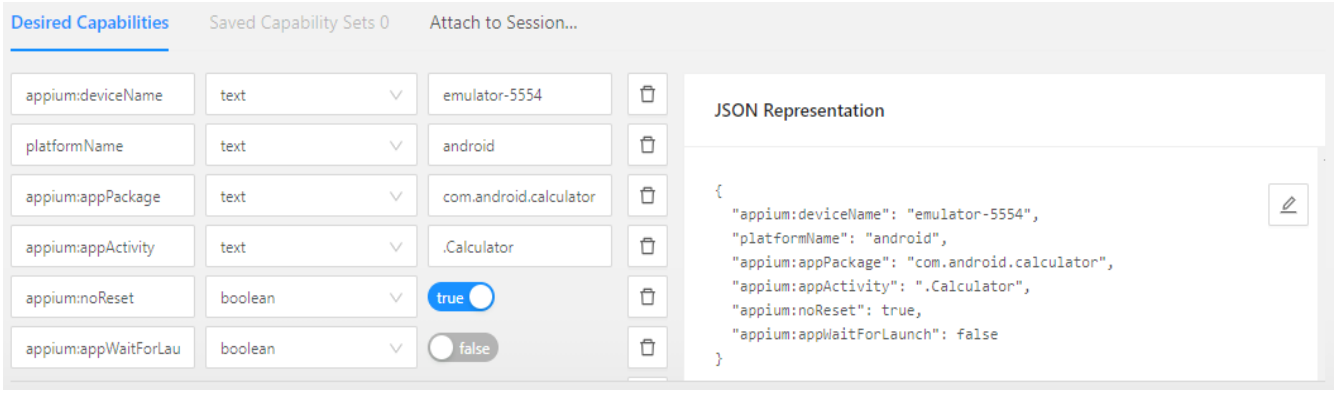


Рис. 2.5. Desired Capabilities Appium

Appium має невеликі відмінності з точки зору виконання, коли справа стосується Android та iOS. Appium на Android використовує фреймворк UIAutomator для автоматизації. UIAutomator – це платформа, створена Android для автоматизації. Отже, давайте подивимося, як саме Appium працює на Android.

Клієнт Appium (C/Java/Python/і т.д.) підключається до сервера Appium і обмінюється даними через дротовий протокол JSON.

Потім Appium Server створює сеанс автоматизації для клієнта, а також перевіряє бажані можливості клієнта. Потім він підключається до відповідних фреймворків, наданих постачальником, наприклад UIAutomator.

Потім UIAutomator зв’яжеться з bootstrap.jar, який працює в симуляторі/емуляторі/реальному пристрої для виконання операцій клієнта. Тут bootstrap.jar відіграє роль TCP-сервера, який ми можемо використовувати для надсилання тестової команди для виконання дії на пристрої Android за допомогою UIAutomator.

Схема архітектури Android Appium нижче дає візуальне представлення вищезазначених кроків.

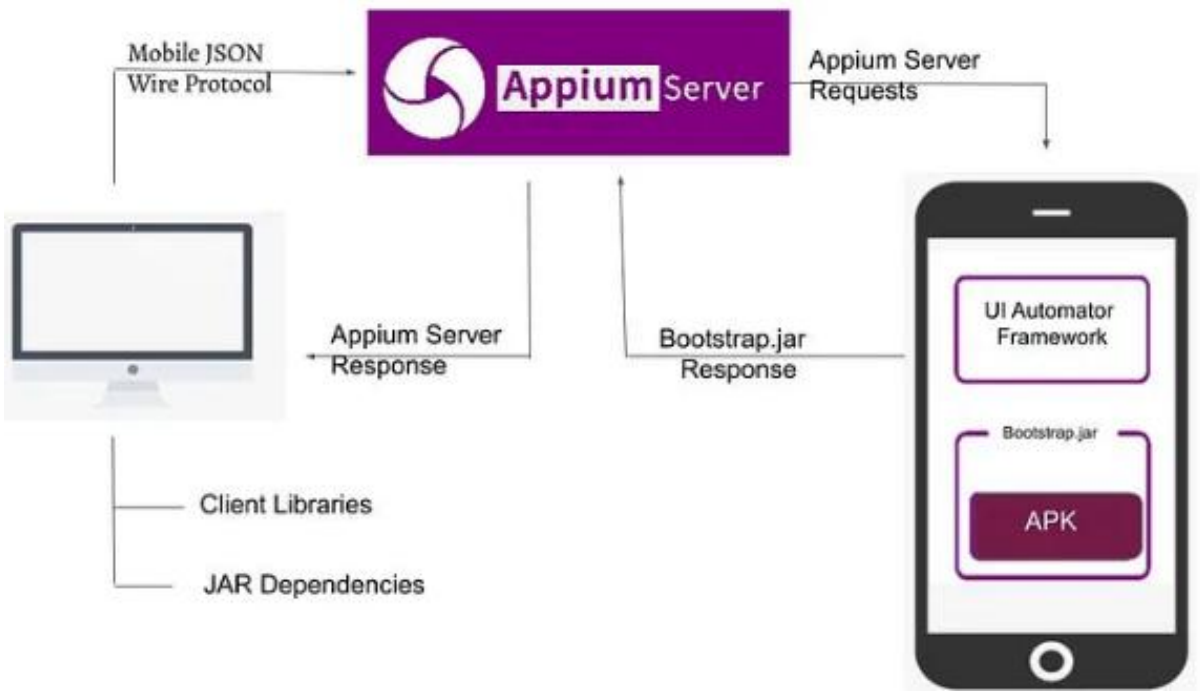


Рис. 2.5. Як працює Appium на Android

Отже, схема роботи Appium на Android показує, що Appium Server є головним компонентом, який керує всіма іншими компонентами та забезпечує автоматизоване тестування мобільних додатків на Android-пристроях.

2.7. Система управління тестами TestRail

Для досягнення ефективного тестування багато проєктів використовують інструменти для створення звітності. Один з таких інструментів – TestRail, який є важливим інструментом для звітування про результати тестування. Завдяки TestRail можна зберігати та відстежувати інформацію про тести, результати їх виконання та виконавців. Такі дані можуть бути корисні для аналізу якості продукту, виявлення недоліків та покращення процесу розробки [6].

TestRail дозволяє створювати тест-кейси та тест-плани, визначати пріоритети для тестів та відстежувати їх виконання. Також можна додавати коментарі до тестів, прикріплювати файли та скріншоти до результатів тестування, щоб зберігати детальну інформацію про виконання тестів.

TestRail також має можливість генерації звітів з результатами тестування, які можна експортувати у різноманітні формати, наприклад, HTML, PDF, Excel тощо. Це дозволяє ділитися інформацією про результати тестування зі всіма учасниками проєкту та широкою громадськістю.

Крім того, TestRail може інтегруватися з іншими інструментами для автоматизованого тестування, такими як Selenium, Jira, Appium, Jenkins тощо, що дозволяє автоматично оновлювати результати тестів у TestRail та автоматично генерувати звіти.

В цілому, TestRail є потужним інструментом управління тестами, який може забезпечити якість продукту та покращити ефективність процесу тестування.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ СИСТЕМИ ТЕСТУВАННЯ

3.1. Інформація про мобільний додаток Udemy

Udemy є онлайн-платформою, яка доступна на вебсайті та мобільних пристроях з операційними системами Android та iOS.

Для викладачів на платформі доступний спеціальний інструмент для створення курсів – Udemy Course Creation Tool. Цей інструмент дозволяє створювати відеоуроки, завдання та додавати інші матеріали до курсу.

Курси на Udemy містять відеоуроки, тексти, завдання, тести та інші ресурси. Студенти можуть вивчати матеріали на будь-якому пристрої з доступом до Інтернету, зокрема на комп'ютерах, планшетах та смартфонах.

Платформа має функцію автоперекладу, яка дозволяє перекладати відеоуроки та інші матеріали на різні мови. Udemy підтримує декілька мов, включаючи англійську, іспанську, французьку та німецьку.

Udemy також має вбудовану систему оцінювання, яка дозволяє студентам оцінювати курси та залишати відгуки про них. Крім того, платформа має систему підтримки, яка допомагає студентам та викладачам розв'язувати будь-які проблеми, що виникають в процесі навчання.

Udemy використовує технології машинного навчання та аналізу даних, щоб забезпечити студентам персоналізований досвід навчання та рекомендації щодо курсів, які можуть їх зацікавити. Крім того, платформа використовує захисні технології для забезпечення безпеки даних користувачів.

3.2. Тестова документація

Алгоритмом реалізації системи тестування можемо вважати STLC, про який згадувалось у першому розділі. Тому завдяки STLC уже відомо, що перед початком автоматизованого тестування потрібно мати належну тестову документацію, яка

допоможе забезпечити ефективність тестування та допоможе знизити кількість помилок в процесі розробки.

Одні з основних типів тестової документації, які потрібні перед початком автоматизованого тестування, це:

- Тестовий випадок/тест-кейс (Test Cases) – документ, який містить послідовність кроків, які необхідно виконати для проведення тестування. Кожен тест-кейс повинен включати опис очікуваних результатів [10]. Саме цей вид документації розкриється в даному пункті.
- Звіти про тестування (Test Reports) – документ, який містить результати тестування, включаючи кількість успішних та неуспішних тестів, опис помилок та їх пріоритет, час виконання тестів та іншу статистику. Звіти про тестування формуються в кінці усього процесу автоматизації. Іншими словами це документ аналізів та висновків [10].
- Документація з тестування (Testing Documentation) – документація, що містить інформацію про стратегії, методики, інструменти та процеси тестування, що використовуються в проєкті [10]. Можна стверджувати, що цей документ описано у другому розділі та частково у першому.

Говорячи про тест-кейси, зазначимо, що вони є не тільки документом, але їх також розглядати як своєрідний алгоритм тестування, оскільки він описує послідовність кроків для перевірки певної функціональності або властивості програмного забезпечення. До того ж тест-кейси пишуться на основі користувацьких історій (User story).

Наприклад, проєкт менеджер надав таку User story для розробки та тестування додатка Udemy:

1. Я, як студент, хочу мати можливість додавати курси в кошик, щоб зберігати їх для подальшого перегляду та купівлі.
2. Я, як студент, хочу мати можливість видаляти курси з кошика, щоб зберігати її в актуальному стані та видаляти зайві курси.

3. Я, як студент, хочу мати можливість переносити курси з кошика до списку бажань, щоб зберігати список потрібних курсів та планувати своє навчання.

4. Коли користувач знаходить курс, який йому цікавий, він може натиснути кнопку "Додати в кошик" для того, щоб додати курс у свій кошик.

5. Коли користувач переходить до кошика, він може переглянути всі курси, які він додав до неї, та видалити з неї ті, які вже не потрібні або перенести їх до списку бажань.

6. Коли користувач переходить до списку бажань, він може переглянути всі курси, які він додав до неї та перенести їх назад у кошик, якщо вирішив придбати їх.

Таким чином, користувач може керувати своїми навчальними планами, зберігати необхідні курси та планувати своє навчання.

На основі цієї User story проведено аналіз вимог та тест-планування, після чого відповідно починається третя фаза STLC. Під час фази тест-дизайну написано кілька наступних тест-кейсів (табл.3.1 та табл.3.2).

1) Ідентифікатор тесту: C98573.

Назва модуля: Кошик.

Пріоритет: Середній (Medium).

Тип: Функціональність (Functionality).

Назва: Верифікація функціональності додавання і видалення курсу з кошика.

Опис: підтвердження того, чи користувач може додати курс до кошика та очистити її.

Таблиця 3.1.

Верифікація функціональності додавання і видалення курсу з кошика

Кроки	Очікуваний результат тестування
Відкрити додаток	Додаток відкрився
Натиснути на іконку Пошуку	Сторінка пошуку відкрилася

Продовження таблиці 3.1.

Кроки	Очікуваний результат тестування
Вибрати будь-який ключовий запит з категорії Топ-пошуку (наприклад, "python")	В поле пошуку введено ключове слово
Вибрати курс_1 зі списку > натиснути на нього	Відкрилася сторінка зі списком курсів Відкрилася сторінка деталей course_1
Натиснути кнопку "Додати до кошика" > кнопку "Продовжити до кошика"	Кошик відкрився
Натиснути на курс у кошику (щоб повернутися до деталей курсу)	Повторно відкрита сторінка деталей course_1
Натиснути кнопку "Додати до списку бажаного" > ОК	Віконце з повідомленням "Adding this course to your wishlist will remove it from your cart" було закрито натисканням на кнопку Ok Кнопка "Add to wishlist" змінилася на кнопку з написом "Wishlisted" Кнопка "Go to cart" змінилася на кнопку з написом "Add to cart"
Натиснути на іконку кошика в правому верхньому куті додатка	Кошик відкрився
Перевірити, що кошик порожній.	Кошик порожній

2) Ідентифікатор тесту: C98574.

   Назва модуля: Кошик.

   Пріоритет: Середній (Medium).

   Тип: Функціональність (Functionality).

   Назва: Верифікація різних маніпуляцій в кошику.

Опис: підтвердження того, чи користувач може проводити різні маніпуляції із курсами в кошику.

Таблиця 3.2.

Верифікація різних маніпуляцій в кошику

Кроки	Очікуваний результат тестування
Відкрити додаток	Додаток відкрито
Натиснути на значок Пошуку	Відкрито сторінку пошуку
Натиснути на будь-яке ключове слово зі списку Категорій (наприклад, Розробка)	Введено ключове слово в поле пошуку
Вибрати перший курс зі списку і натиснути на нього	Відкрито список курсів
Натиснути кнопку «Додати до списку бажаного»	Відкрито сторінку деталей курсу 1
Повернутися до всіх курсів на вкладці Пошуку	Кнопка "Додати до списку бажань" змінена на "В списку бажань"
Вибрати другий курс зі списку і натиснути на нього	Відкрито список курсів на вкладці Пошук
Натиснути кнопку «Додати до кошика»	Відкрито сторінку деталей курсу_2
Натиснути кнопку «Продовжити» на випливаючому вікні з рекомендованими курсами	Кошик відкрито
Натиснути кнопку «Видалити» під курсом_1 з розділу «Список бажаного»	Повідомлення "Видалено" відображається. Курс з розділу Список бажань видалено

Продовження таблиці 3.2

Кроки	Очікуваний результат тестування
Натиснути кнопку «Зберегти на потім» під курсом_2 з кошика	Повідомлення "Збережено" відображається. Курс з розділу Елементи видалено та збережено в розділі "Збережені для пізніше".
Натиснути кнопку «Видалити» під курсом з розділу «Збережено на потім»	Повідомлення "Видалено" відображається. Курс_2 з розділу "Збережені для пізніше" видалено

Після того, як створено тестову документацію, а саме тест-дизайн, на основі якого створюються автоматизовані тести, можемо переходити до підготовки середовища розробки.

3.3. Системні вимоги

Для того, щоб середовище розробки працювало коректно і для забезпечення правильної та стійкої роботи навчального додатка розроблено вимоги до пристрою, на якому буде відбуватися тестування:

- Операційна система: Windows 10, macOS Catalina або вище, або Linux (64-бітна).
- Процесор: Intel Core i5 або аналогічний з тактовою частотою не менше 2.5 ГГц.
- Оперативна пам'ять: не менше 8 ГБ.
- Відеокарта: мінімум NVIDIA GeForce GTX 1050 або аналогічна.
- Диск: не менше 5 ГБ вільного місця на жорсткому диску або SSD. Рекомендація: для роботи з емулятором краще мати близько 20 Гб вільного місця на жорсткому диску або SSD.
- Python: версія 3.6 або вище.

- Бібліотеки Python: Appium-Python-Client, Pytest, Selenium.
- Інтернет: доступ до інтернету зі швидкістю не менше 10 Мбіт/с.

То ж коли всі вимоги виконані можна братися до налаштування середовища та власне розробки самої системи.

3.4. Процес розробки системи тестування

Для того, щоб почати розробку системи тестування, а саме автоматизованого тестування, потрібно встановити всі необхідні програми та налаштувати їх. Для роботи обрано програми та інструменти, які описані в другому розділі.

Перш за все після встановлення Appium відкриваємо програму та за допомогою програми-інтерфейсу запускаємо сервер. При запуску сервера Appium вказуємо параметри, такі як IP-адреса, порт і параметри проксі-сервера.

Після цього підключаємо фізичний пристрій до комп'ютера за допомогою USB-кабелю і дозволити на ньому режим розробника та USB-налагодження. Якщо ж використовувати емулятор, то потрібно буде налаштувати його у програмі AVD manager, про який писалось у другому розділі. І саме на цьому етапі, потрібно встановити Android SDK – для того, щоб були налаштовані всі драйвери, необхідні для роботи. Попри те, що Android Studio також можна використовувати для роботи з фізичним пристроєм, для перегляду додатка та проведення усіх тестів у даній кваліфікаційній роботі використано Vysor та встановлено додаток Udemy (для роботи з емулятором завантажуємо apk додаток). Таким чином розробник має доступ до всіх компонентів мобільного додатка.

Щоб здійснити підключення до пристрою Android та відповідно створити сесію, потрібно ще налаштувати параметри підключення в інспекторі Appium. Ці параметри можна налаштувати в об'єкті DesiredCapabilities. Використано такі параметри, як platformName, deviceName, appPackage, appActivity, noReset (Рис.3.1). Щоб дізнатись атрибути для deviceName, appPackage, appActivity використано командний рядок із такими командами, як:

- adb devices
- adb logcat > logfile.txt – створюється файл, у якому за допомогою пошукового слова “for activity” знаходимо необхідні дані

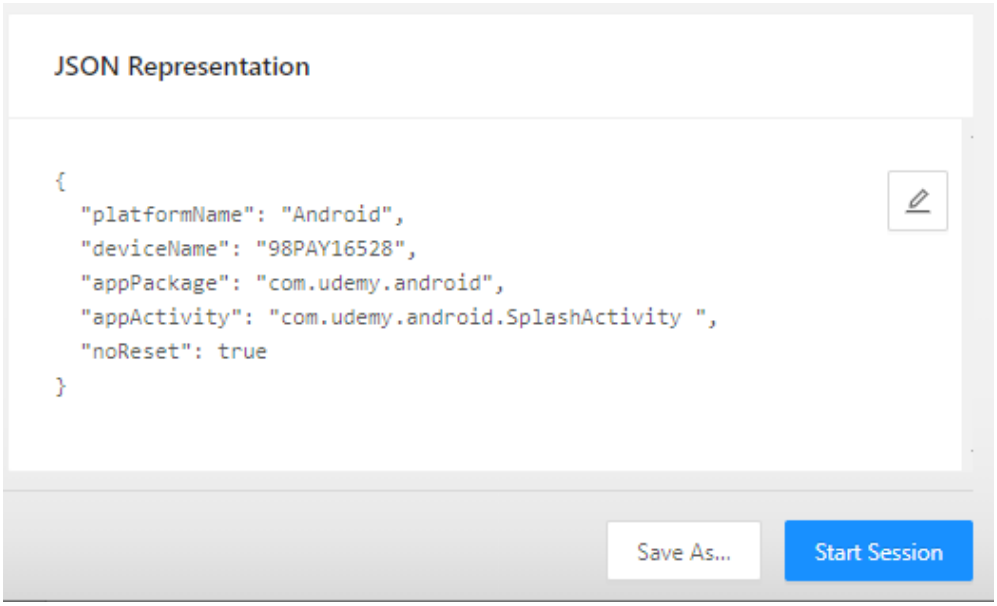


Рис. 3.1. JSON код, створений із DesiredCapabilities

Після запуску сервера Appium і настройки параметрів підключення потрібно створити нову сесію, яка дозволяє здійснювати взаємодію між сервером і пристроєм. Для створення сесії використовують метод webdriver.Remote().

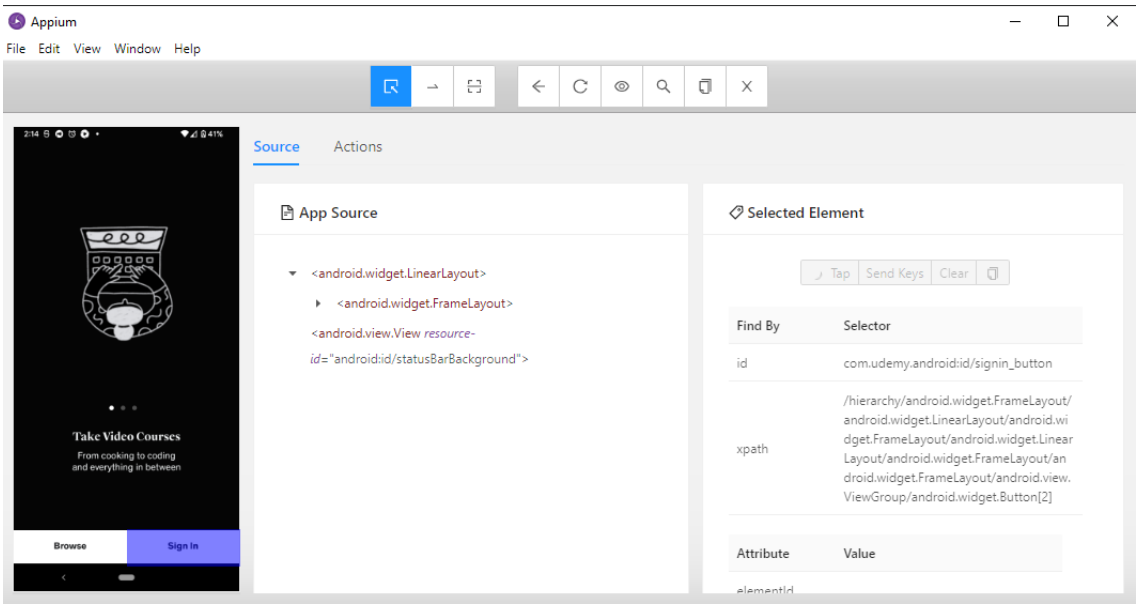


Рис. 3.2. Вікно сесії Appium мобільного додатка

Нижче наведений код налаштовує параметри desired_capabilities для Android-пристрою, створює драйвер Appium, запускає додаток та дозволяє працювати з додатком уже в середовищі розробки Pycharm.

```
from appium import webdriver
        def setUp(self):
desired_capabilities = {
   'platformName': 'Android',
   'deviceName': '98PAY16528',  # if emulator – use ‘deviceName’: ‘emulator-5554’
   'appPackage': ' com.udemy.android',
   'appActivity': ' com.udemy.android.SplashActivity '
   'noReset': True
}
self.driver=webdriver.Remote("http://localhost:4723/wd/hub", desired_capabilities)
        def tearDown(self):
         self.driver.quit()
```

Наступним кроком після підключення до пристрою та створення сесії є написання автоматизованих тестів, користуючись розробленими тестовими випадками. Виконання тестів можна здійснити за допомогою фреймворку Pytest та зазвичай воно відбувається в циклі, де кожен крок тестування передбачає взаємодію з додатком та перевірку результату. Наприклад, використано методи класу webdriver, такі як click(), щоб клікнути на елемент, або send_keys(), щоб ввести текст. Також використано метод find_element_by_id() для знаходження елемента за його унікальним ідентифікатором (ID). Адже кожен елемент на сторінці має свій унікальний ідентифікатор, який можна використовувати для знаходження цього елемента. Після закінчення роботи з додатком потрібно викликати метод quit() для закриття драйвера Appium, оскільки це важливо для того, щоб не залишати відкритих сесій браузера на сервері, що може призвести до проблем з пам'яттю та

іншими проблемами. Розроблені автоматизовані тестові випадки можна оглянути в Додатку В.

3.5. Використання програмної системи

Для того, щоб використовувати новостворену систему гарантування потрібно виконати ряд певних дій, а саме:

1) Щоб відкрити потрібний проєкт – відкриваємо середовище розробки і на головній панелі можемо одразу ж обрати наш проєкт з тестами або через панель інструментів, обравши вкладку File > Open, з дерева папок обираємо ту, де розміщений наш проєкт.

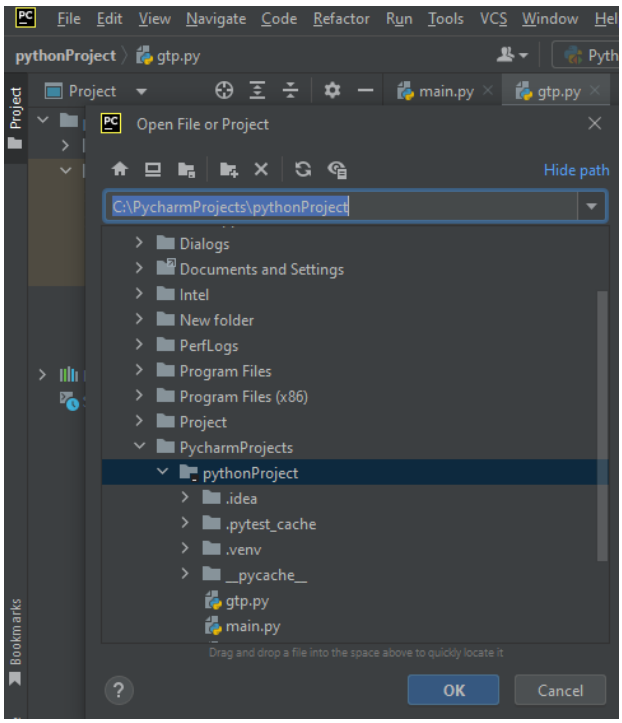


Рис. 3.3. Дерево папок з необхідним проектом

2) Наступне, що потрібно зробити – це під’єднати необхідний фізичний пристрій до комп'ютера за допомогою USB-кабелю та стартувати роботу сервера Appium, встановити всі DesiredCapabilities. Після чого розпочинаємо користувацьку сесію Appium.

3) Після того, як зроблено з'єднання між Pycharm, Appium та фізичним пристроєм – запускаємо проєкт та розпочинаємо процес тестування. Першим і, певно, найбільш важливим тестом вважається тест на перевірку відкриття програми.



Рис. 3.4. Відображення успішно пройденого першого тесту

Якщо вже говорити про сам процес тестування, то варто згадати, що тести можуть як проходити перевірку, так і не проходити. Їхній статус можна побачити в консолі.

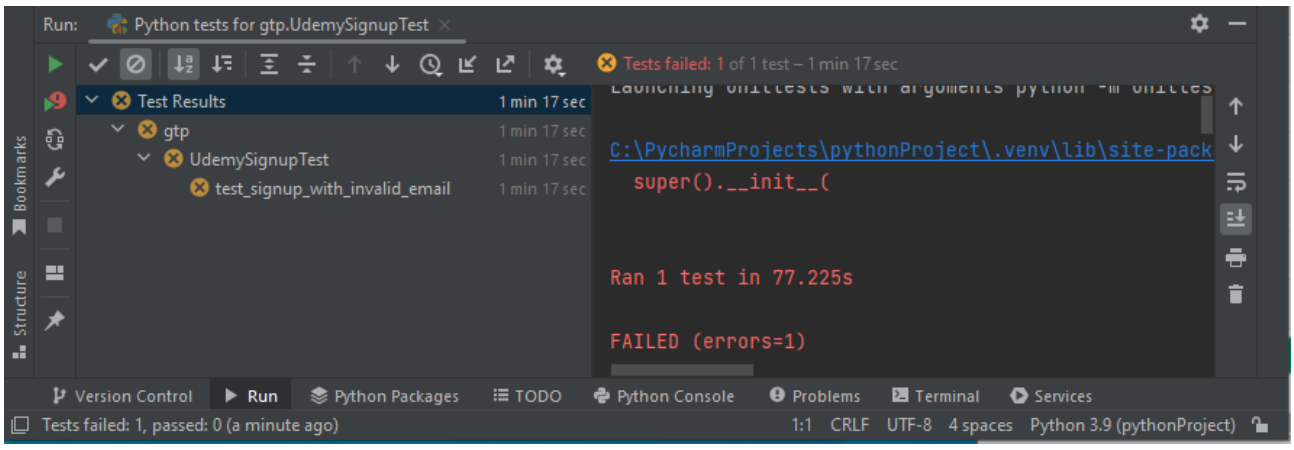


Рис. 3.5. Повідомлення про помилку при проходженні тесту

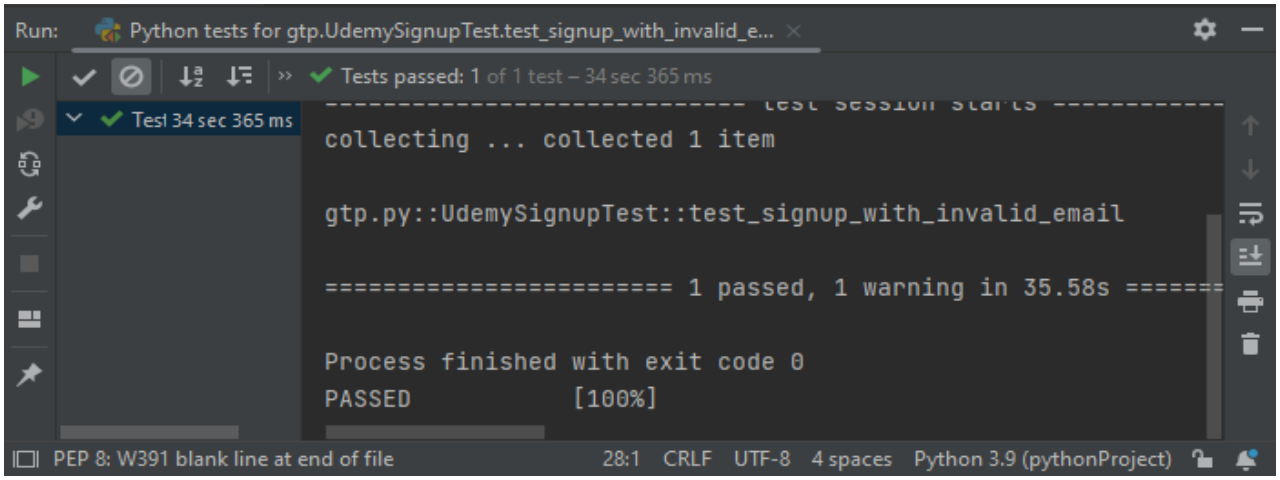


Рис. 3.6. Повідомлення про успішне проходження тесту

На кінець, варто згадати, що створюються нові версії додатку, відповідно через це деякі тести можуть виконуватись неуспішно. В такому випадку виникає необхідність редагувати існуючий проєкт або додати нові дані, а для цього потрібно відтворювати всі ті ж дії, що описані при початковій реалізації проєкту.

3.6. Аналіз отриманих результатів

Аналіз результатів автоматизованого тестування допомагає виявляти проблеми та вдосконалювати якість програмного продукту.

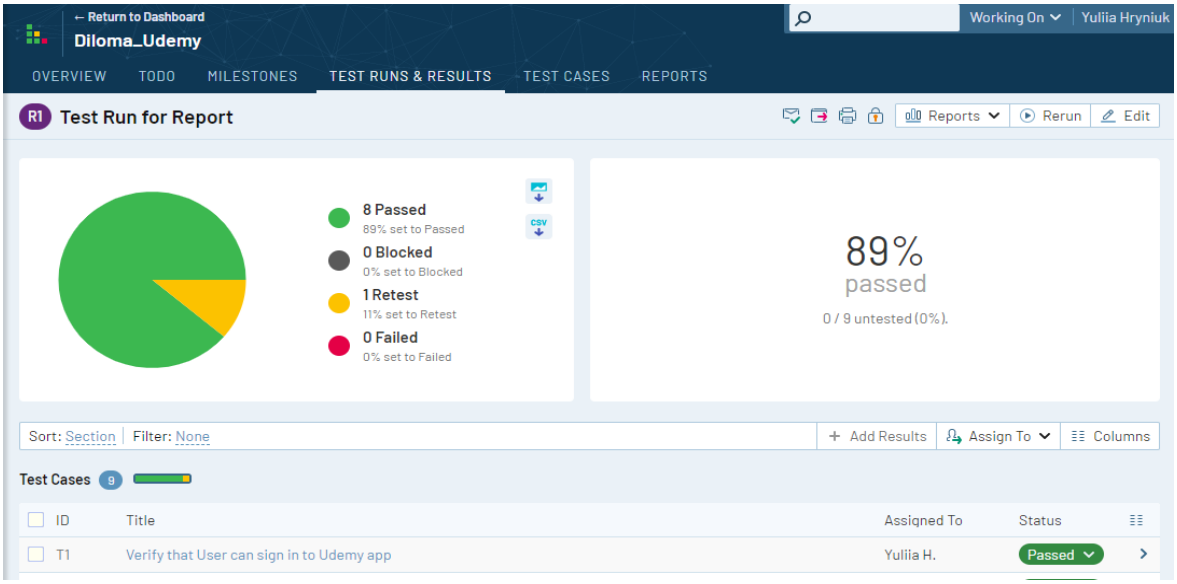


Рис. 3.7. Отримані результати після закінчення процесу тестування

Для аналізу результатів створеної системи використано систему управління тестами TestRail. Завдяки зручному графічному та аналітичному відображенні кількості пройдених тест-кейсів, можна чітко побачити, чи виконала розроблена система свою функцію, чи ні. Відповідно до показників 1 тест-кейс має статус Retest, а отже, його потрібно перетестувати. Дана поведінка пов’язана зі змінами у програмному коді одного тесту, тому необхідно переконатись, що автоматизований тест працює коректно та немає дефекту. І тільки, коли 100% тест-кейсів буде пройдено, можна стверджувати, що система гарантування якості досягла своєї мети. Але і без 100% пройдених тестів, можемо стверджувати, що частково автоматизована система виконала свою роботу. Адже час проходження одного тест-кейсу скоротився у 4.8 рази (Табл.3.3).

Таблиця 3.3.

Час проходження тест-кейсу

	Ручне тестування	**Автоматизоване тестування**
Середній час	4 хв (240 с)	50 с

Відповідно збільшення кількості тестів може допомогти виявити помилки, що дозволить покращити якість продукту та зменшити ризики випуску продукту з помилками. Також автоматизація тестування може зменшити час процесу тестування [25].

3.7. Удосконалення автоматизованого тестування та системи гарантування якості в цілому

Існує безліч можливостей для розвитку автоматизованого тестування та системи гарантування якості в цілому. І всі вони досить широкі і залежать від розвитку технологій та вимог ринку. Деякі з можливих перспектив та напрямів розвитку включають:

1) Використання штучного інтелекту та машинного навчання: за допомогою цих технологій можна розширити можливості автоматизованого тестування, зокрема, виявляти та запобігати помилкам, робити більш детальний аналіз результатів тестів та надавати рекомендації щодо покращення якості продукту.

2) Розширення покриття тестами: збільшення кількості тестів та їх покриття дозволяє виявляти помилки на ранніх стадіях та запобігати їх поширенню у продукті.

3) Інтеграція з CI/CD системами: автоматичне виконання тестів при кожному зміні в коді дозволяє виявляти та виправляти помилки швидше та забезпечує стабільність продукту [19].

4) Використання хмарних рішень: використання хмарних інфраструктур та рішень дозволяє зменшити витрати на обладнання та програмне забезпечення для тестування, забезпечує високу доступність та масштабованість тестування.

5) Використання контейнеризації: контейнеризація тестового середовища дозволяє легко та швидко розгортати тестові середовища та забезпечує стабільність тестів незалежно від середовища виконання.

6) Баланс між ручним та автоматизованим тестуванням: комбінація цих двох методів приведе до створення якіснішого продукту. З одного боку автоматизація пришвидшує виконання тестів, з іншого боку – якщо тести не належним чином плануються та розробляються, може виникнути ситуація, коли витрати на розробку тестів перевищують користь від їх виконання [25]. Або ж, якщо автоматичні тести забирають більше часу, ніж ручні тести, це може збільшити загальний час тестування і тоді автоматизоване тестування стане неефективним.

І це лише декілька з можливих напрямків розвитку системи гарантування якості та автоматизованого тестування. Кожна компанія та команда має свої особливості та вимоги, тому важливо постійно працювати над удосконаленням [24].

ВИСНОВКИ

В результаті виконання дипломної роботи була розроблена та реалізована система гарантування якості, яка базується на автоматизованому тестуванні. Завдяки цій системі було можливо автоматизувати процес тестування, зменшити час, необхідний для виявлення та виправлення помилок, та забезпечити вищу якість додатка.

У результаті виконання автоматизованих тестів було підтверджено якісну роботу додатка Udemy. Також результати дослідження свідчать про те, що автоматизоване тестування може бути ефективним інструментом для забезпечення якості програмного забезпечення та його покращення.

Додатково хочу звернути увагу на вибір правильних інструментів для автоматизації тестування мобільного додатку, оскільки невірно обраний інструмент може призвести до непотрібних витрат часу і коштів, а також створена система не була б такою ефективною, як вона є. Саме тому для забезпечення високої якості додатку потрібно мати надійні, ефективні інструменти для автоматизованого тестування.

Окрім вибору інструментів автоматизованого тестування, дуже важливо мати гарну тестову документацію. Це допоможе зрозуміти, які тести потрібно запускати, від яких функцій залежать ті чи інші тести, які тести вже запускалися та який був результат і т.д. Тестова документація дозволяє зберігати інформацію про тести протягом всього життєвого циклу додатка та робити її доступною для всіх учасників процесу розробки. Це сприяє більш ефективному тестуванню, швидшому виявленню дефектів та швидкому вирішенню проблем.

І останній фактор, на який звертається мало уваги, але він надзвичайно важливий – це системні вимоги. Не дарма про них розписано у даній роботі. Наприклад, якщо буде процесор Intel Pentium з тактовою частотою 2.16ГГц, то використання емулятора буде неможливим. Тому врахування системних вимог є

важливою складовою при розробці програмного забезпечення, оскільки це дозволяє розробникам програмного забезпечення зосередитись на функціональності та можливостях програми, замість того, щоб витрачати час на розв'язання проблем, пов'язаних з недостатнім обсягом пам'яті чи недостатньою швидкістю процесора.

Отже, можна стверджувати, що розроблена система гарантування якості з використанням автоматизованого тестування була успішно впроваджена та дозволяє покращити якість мобільного додатку Udemy. І в той самий час, як і кожна програма чи система, розрозроблена та реалізована система гарантування якості може і повинна удосконалюватись. Адже удосконалення допомагає досягти наступних цілей:

1. Підвищення задоволеності користувачів: якщо додаток працює стабільно, зручно та не має помилок, користувачі будуть задоволені та з радістю будуть використовувати його.
2. Зменшення витрат на розробку та тестування додатку: це досягається завдяки тому, що помилки виявляються на ранніх етапах розробки, коли виправлення коштує менше, ніж після релізу додатка.
3. Збільшення впевненості в продукті: система гарантування якості допомагає переконатися в тому, що додаток працює належним чином та відповідає вимогам.
4. Збільшення конкурентоспроможності: добре протестований та якісний додаток привертає більше користувачів та підвищує конкурентоспроможність на ринку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Гончарук С., Гончарук О. Методології тестування програмного забезпечення. Київ: Центр учбової літератури, 2016. 288 с.

2. Ковальов В., Трохимець О. Тестування програмного забезпечення: організація, техніка, інструментарій. Київ: Кондор, 2015. 336 с.

3. Крамаренко В., Менделєєв О., Королева Н. Якість програмного забезпечення. Аспекти тестування. Київ: Центр учбової літератури, 2017. 320 с.

4. Мельник А., Поліщук О. Якість програмного забезпечення. Харків: Видавництво НТУ "ХПІ", 2017. 344 с.

5. Соловйов О. Якість програмного забезпечення: підходи, методи, технології. Київ: Навчальна книга – Богдан, 2016. 384 с.

6. Al Shalloway, Scott Bain, Ken Pugh. Lean-Agile Software Development: Achieving Enterprise Agility. Addison-Wesley Professional, 2019. 192 p.

7. Andreas Spillner, Tilo Linz, and Thomas Rossner. Software testing foundations: A study guide for the certified tester exam. Rocky Nook, 2014. 320 p.

8. Appium. URL: https://appium.io/

9. Brian Okken and Alexander Hultnäs. Python Testing with pytest. Pragmatic Bookshelf, 2017. 240 p.

10. Cem Kaner, James Bach, and Bret Pettichord. Lessons learned in software testing: A context-driven approach. Wiley, 2015. 320 p.

11. Eric Ries. The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses. Currency, 2011. 336 p.

12. Eric Matthes. Python Crash Course: A Hands-On, Project-Based Introduction to Programming. No Starch Press, 2019. 544 p.

13. Erik van Veenendaal. Testing in the digital age: AI makes the difference. UTN Publishers, 2021. 376 p.

14. Jake Knapp, John Zeratsky, and Braden Kowitz. Sprint: How to Solve Big Problems and Test New Ideas in Just Five Days. Simon & Schuster, 2016. 288 p.

15. Jennifer Greene and Andrew Stellman. Head First Agile: A Brain-Friendly Guide to Agile and the PMI-ACP Certification. O'Reilly Media, 2017. 490 p.

16. John D. Musa, Anthony I. C. Chen, and Tommie G. Kim. Software reliability: Measurement, prediction, application. John Wiley & Sons, 2015. 656 p.

17. Ken Schwaber and Jeff Sutherland. The Scrum Guide. URL: https://www.scrum.org/resources/scrum-guide.

18. Mark Winteringham. Python for Testers: Learn to Program by Building a Test Automation Framework. Leanpub, 2017. 347 p.

19. Martin Fowler. Continuous Integration. URL: https://martinfowler.com/articles/continuousIntegration.html.

20. Martin Pol and Ruud Teunissen. The Little Black Book of Project Management. Van Haren Publishing, 2018. 200 p.

21. Michael Nir. Agile Product Management: Product Owner: 27 Tips To Manage Your Product And Work With Scrum Teams. Sapir Consulting, 2014. 92 p.

22. Rex Black. Managing the testing process: Practical tools and techniques for managing hardware and software testing, 3rd edition. Wiley, 2015. 648 p.

23. Robert C. Martin. Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall, 2017. 464 p.

24. Roman Pichler. Agile Product Management with Scrum: Creating Products that Customers Love. Addison-Wesley Professional, 2010. 160 p.

25. William E. Lewis. Software testing and continuous quality improvement. CRC Press, 2018. 236 p.

ДОДАТКИ

Додаток А

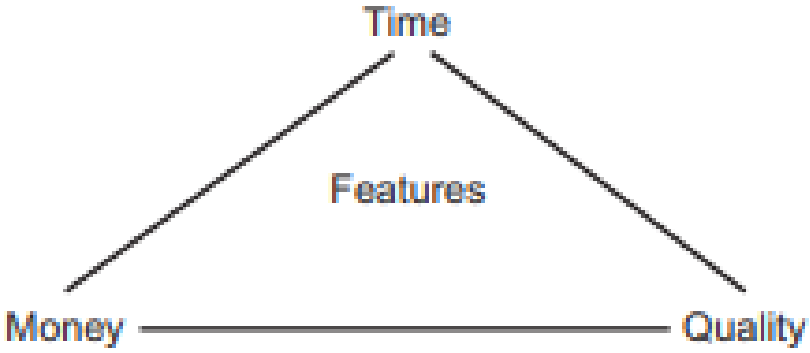


Рис. А.1. Залежність часу, грошей та якості для розробки програми

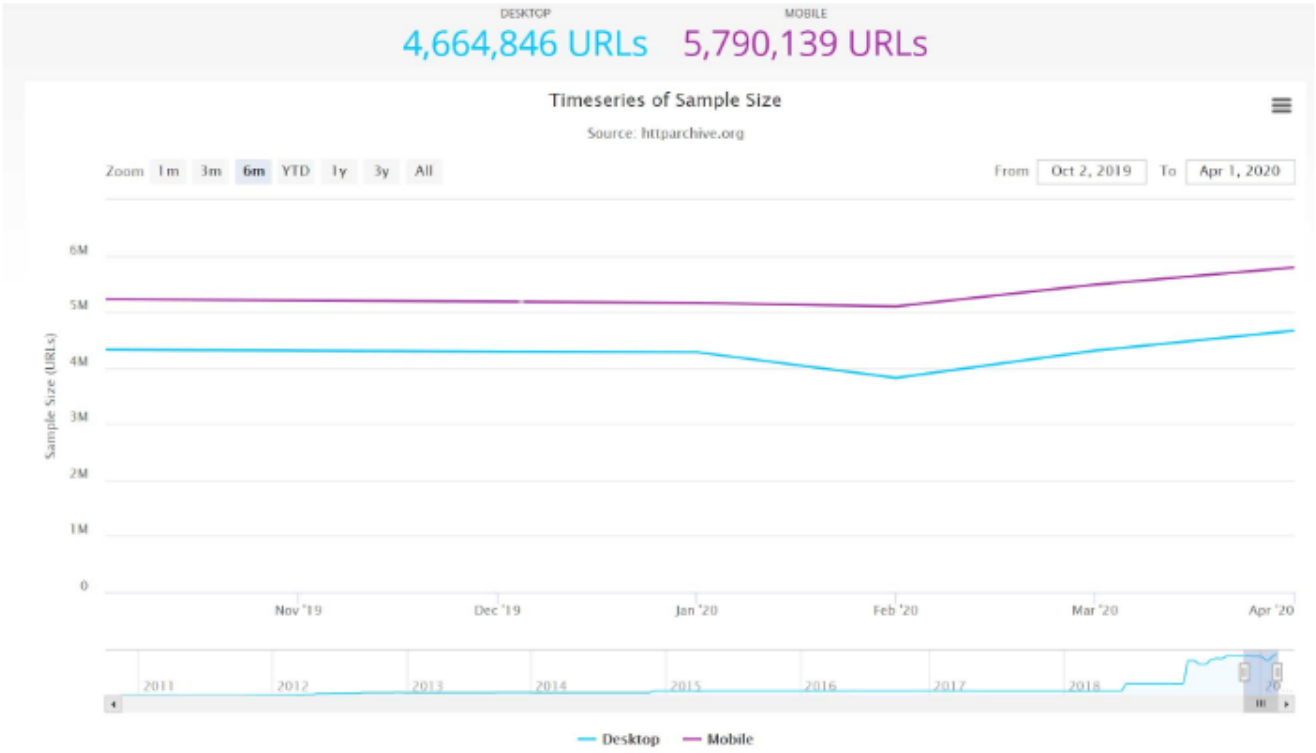


Рис. А.2. Статистика використання мобільних пристроїв

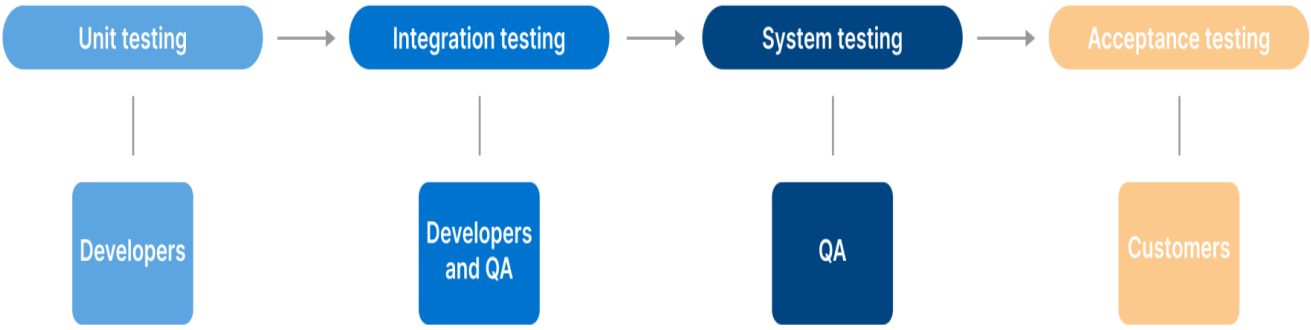


Рис. А.3. Спеціалісти, що відповідають за кожен рівень тестування

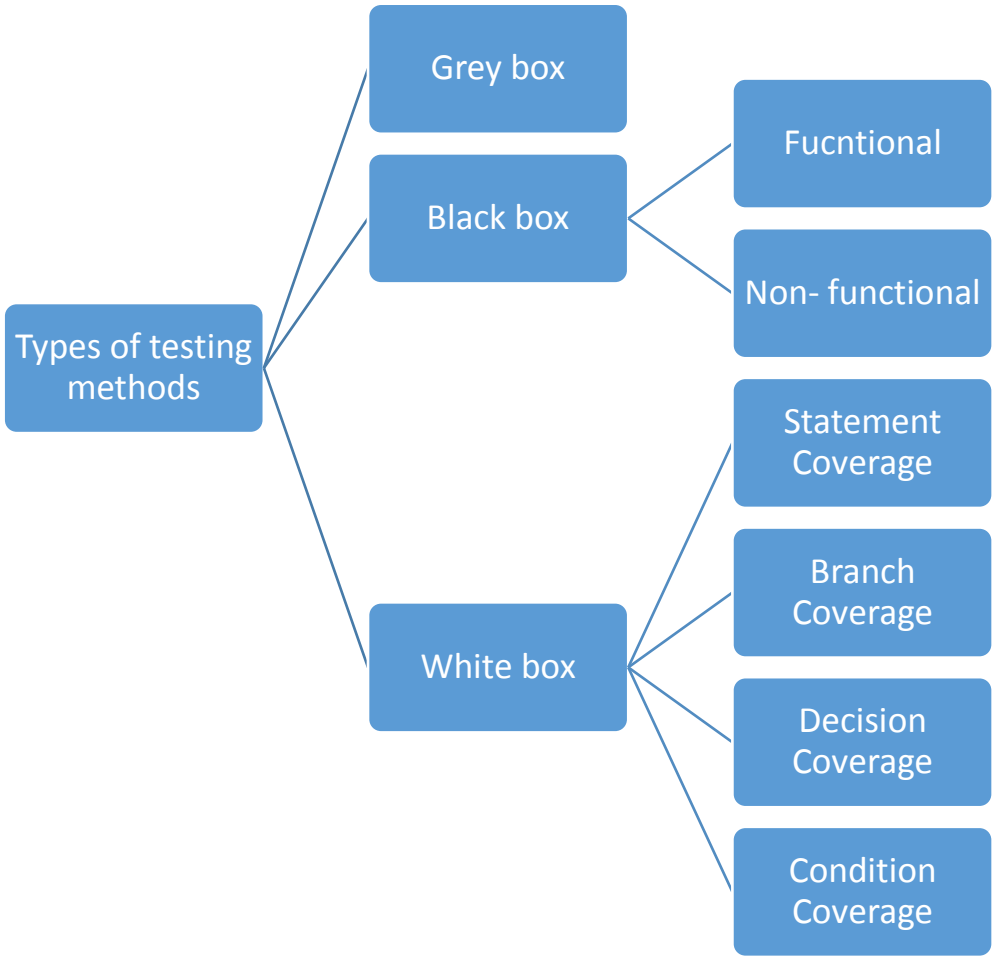


Рис. А.4. Види тестування, що походять від методів тестування

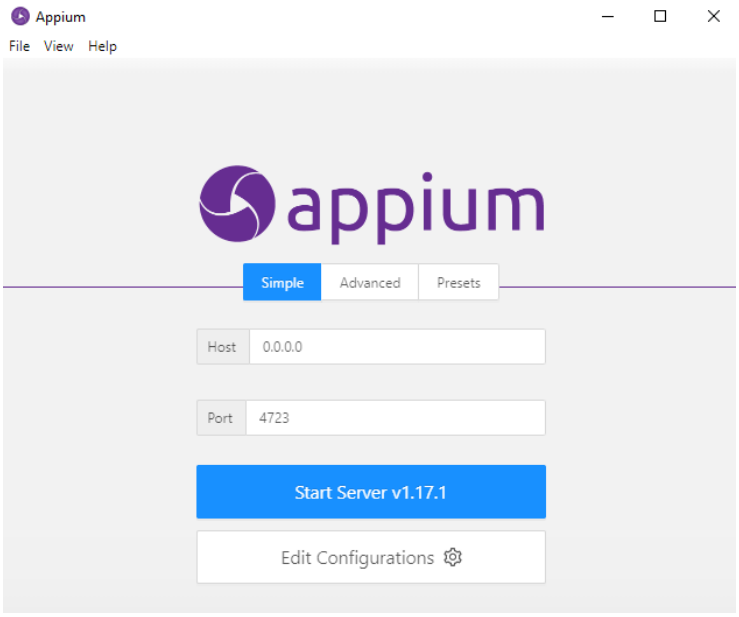


Рис. А.5. Головне вікно Apium перед запуском сервера

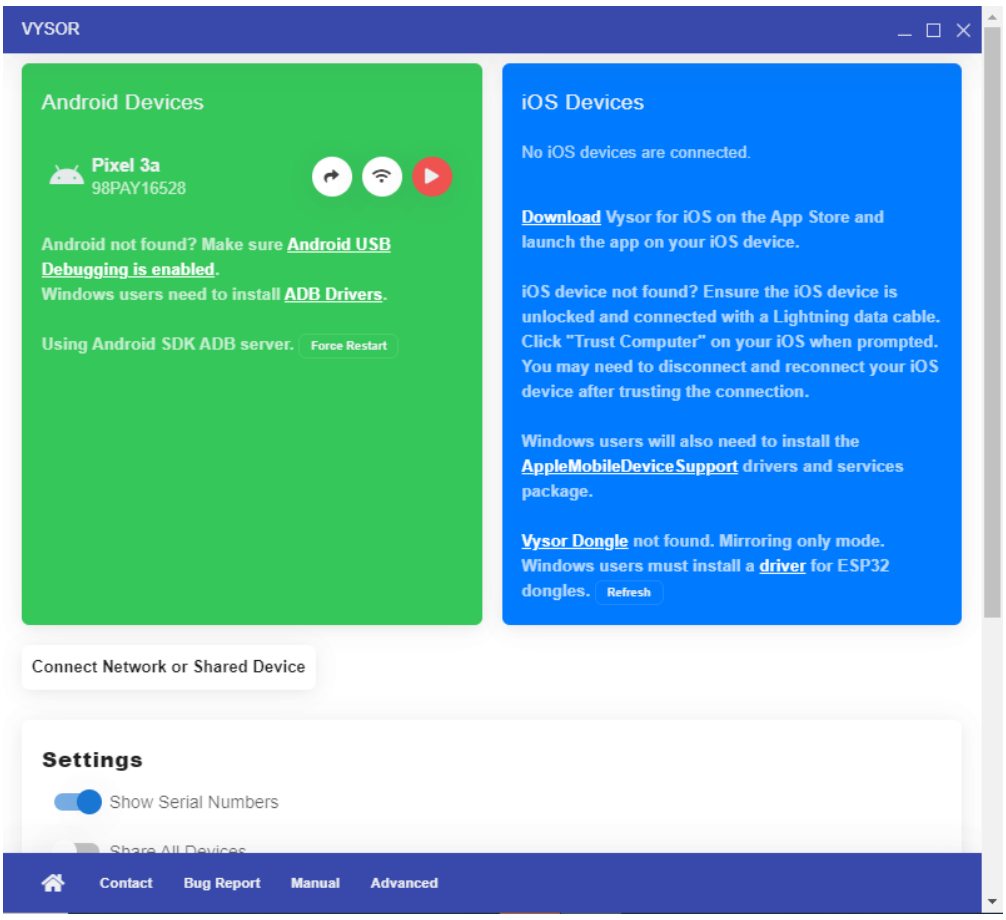


Рис. А.6. Головне вікно Vysor при підключенні до фізичного пристрою

Додаток Б

Таблиця Б.1.

Порівняння рівнів та типів тестування

Рівні тестування	Типи тестування
Рівні тестування програмного забезпечення – це етапи або фази тестування програмного забезпечення, доки воно не досягне кінцевого користувача.	Типи тестування програмного забезпечення – це різні способи тестування програмного забезпечення.
Приклади: модульне тестування, тестування компонентів, інтеграційне тестування, системне тестування та тестування на прийнятність програмного забезпечення, доки воно не досягне кінцевого користувача.	Приклади - типи тестування чорного ящика, білого ящика, сірого ящика.
У сукупності рівні тестування програмного забезпечення охоплюють тестування програмного забезпечення на кожному етапі розробки. Під час модульного тестування перевіряються невеликі пристрої та їх функціональні можливості, під час інтеграції та системної інтеграції програмне забезпечення перевіряється на відповідність вимогам програмного забезпечення. У UAT програмне забезпечення перевіряється з точки зору кінцевого користувача.	У поєднанні типи тестування програмного забезпечення надають різні способи тестування різних функцій і аспектів програмного забезпечення. Проводячи функціональне тестування, ми гарантуємо, що програмне забезпечення відповідає поставленим вимогам, виконуючи нефункціональне тестування, ми перевіряємо програмне забезпечення, забезпечуючи відповідний досвід для користувача.

Продовження додатка Б

Продовження таблиці Б.1

Наприклад, на рівні тестування модуля або компонента розробник може мати план виконання тестування білої скриньки для покращення структури кодування, а також може виконувати функціональне тестування чорної скриньки для перевірки функціональності на рівні кожного модуля.	Загалом, тестувальник виконує різні типи тестування в межах одного рівня тесту. Рівень тестування може містити типи тестування програмного забезпечення білої та/або чорної скриньки та/або сірої скриньки для ретельного тестування системи
Критерії входу. Критерії входу для кожного рівня – це завершення та успішне проходження тестування попереднього рівня. Для рівня модульного тестування критерієм входу є завершення розробки.	Критерії входу. Критерії вступу для кожного типу тесту можуть бути різними та, як правило, залежать від вимог бізнесу.
Критерії виходу. Критерії виходу для кожного рівня тестування програмного забезпечення, як правило, полягають у тому, що всі тестові приклади кожного запланованого типу тестування виконуються та показують сприятливі результати, щоб переконатися, що програмне забезпечення готове до наступного рівня тестування.	Критерії виходу. Необхідно визначити критерії виходу для кожного типу тестування програмного забезпечення, і, як правило, усі тестові приклади певного типу тесту виконуються та дають сприятливі результати.

Продовження додатка Б

Таблиця Б.2.

Випадки, у яких краще використовувати фізичний пристрій чи емулятори

Що тестувати	Справжній пристрій	Емулятори/Симулятори
Функціональне тестування для великих збірок інтеграції		+
Тестування макета UI	+	+
Мобільне веб-тестування	+	+
Тестування на сумісність	+	
Ручне/інтерактивне тестування на фізичних пристроях + жести	+	
Тестування пристрою/системи	+	
Бета-тестування	+	
Моніторинг помилок і звітування	+	
Залежності від апаратного забезпечення (ЦП, GPS тощо)	+	
Тестування дисплея (пікселі, роздільна здатність)	+	
Макет камери	+	
Push-повідомлення (реальні сервіси)	+	

Додаток В

Код програми

```
import pytest
from appium import webdriver
from time import sleep
from appium.webdriver.common.touch_action import TouchAction

@pytest.fixture()
def driver():
    desired_capabilities = {
        'platformName': 'Android',
        'deviceName': '98PAY16528',
        'appPackage': 'com.udemy.android',
        'appActivity': 'com.udemy.android.SplashActivity',
        'noReset': True
    }
    driver = webdriver.Remote("http://localhost:4723/wd/hub", desired_capabilities)
    yield driver
    driver.quit()

#First test-case
def test_sign_in_with_google(driver):
    sleep(5)
    signin_button = driver.find_element_by_id("com.udemy.android:id/sign_in_button")
    signin_button.click()
    sleep(2)
```

```
        signin_with_google_button =
driver.find_element_by_id("com.udemy.android:id/google_login_button")
        signin_with_google_button.click()
        sleep(2)
        driver.press_keycode(4)
        sleep(2)
        close_button =
driver.find_element_by_id("com.udemy.android:id/auth_toolbar_close")
        close_button.click()

    #Second test-case
    def test_create_account(driver):
        sign_in_button =
driver.find_element_by_id("com.udemy.android:id/button_sign_in")
        sign_in_button.click()
        create_account_button =
driver.find_element_by_id("com.udemy.android:id/create_account_button")
        create_account_button.click()
        sign_up_with_email_button =
driver.find_element_by_id("com.udemy.android:id/signup_with_email_button")
        sign_up_with_email_button.click()
        name_field = driver.find_element_by_id("com.udemy.android:id/et_name")
        name_field.send_keys("Test")
        email_field = driver.find_element_by_id("com.udemy.android:id/et_email")
        email_field.send_keys("test@test.t")
        password_field =
driver.find_element_by_id("com.udemy.android:id/et_password")
```

```
        password_field.send_keys("testTEST")
        create_account_button =
driver.find_element_by_id("com.udemy.android:id/create_account_button")
        create_account_button.click()
        error_popup = driver.find_element_by_id("com.udemy.android:id/dialog_title")
        assert error_popup.text == "There was a problem creating your account"
        ok_button = driver.find_element_by_id("com.udemy.android:id/btn_ok")
        ok_button.click()
        email_field.clear()
        ok_button = driver.find_element_by_id("com.udemy.android:id/btn_ok")
        ok_button.click()
        warning_text =
driver.find_element_by_id("com.udemy.android:id/textinput_error")
        assert warning_text.text == "This email address is not valid"
        email_field = driver.find_element_by_id("com.udemy.android:id/et_email")
        email_field.send_keys("auto.testing541@gmail.com")
        create_account_button =
driver.find_element_by_id("com.udemy.android:id/create_account_button")
        create_account_button.click()
        welcome_title =
driver.find_element_by_id("com.udemy.android:id/welcome_title")
        assert welcome_title.text == "Welcome!"


    #Third test-case
    def test_add_to_wishlist(driver):
        search_icon =
driver.find_element_by_id("com.udemy.android:id/toolbar_search_icon")
```

```
        search_icon.click()
        top_searches=driver.find_element_by_id("com.udemy.android:id/top_searches_recycler_view")
keyword=top_searches.find_elements_by_class_name("android.widget.TextView")[0]
            keyword.click()
        courses=driver.find_element_by_id("com.udemy.android:id/courses_recycler_view")
            course = courses.find_elements_by_class_name("android.view.ViewGroup")[0]
            course.click()
            wishlist_btn = driver.find_element_by_id("com.udemy.android:id/add_to_wishlist")
            wishlist_btn.click()
            wishlisted_btn= river.find_element_by_id("com.udemy.android:id/wishlisted")
            assert wishlisted_btn.is_displayed()
            driver.back()
            driver.back()
        wishlist_icon=driver.find_element_by_id("com.udemy.android:id/bottom_navigation_wishlist")
            wishlist_icon.click()
        wishlist_courses=driver.find_element_by_id("com.udemy.android:id/courses_recycler_view")
         assert len(wishlist_courses.find_elements_by_class_name("android.view.ViewGroup")) == 1

        #Fourth test-case
        def test_add_course_to_wishlist(driver):
```

```
        search_icon =
driver.find_element_by_id("com.udemy.android:id/search_button")
        search_icon.click()
        top_searches =
driver.find_elements_by_id("com.udemy.android:id/top_search")
        top_searches[0].click()
        courses =
driver.find_elements_by_id("com.udemy.android:id/browse_course_card")
        courses[0].click()
        wishlist_button =
driver.find_element_by_id("com.udemy.android:id/course_wishlist_button")
        wishlist_button.click()
        wishlisted_button =
driver.find_element_by_id("com.udemy.android:id/course_wishlisted_button")
        assert wishlisted_button.is_displayed()
        driver.back()
        driver.back()
        wishlist_icon =
driver.find_element_by_id("com.udemy.android:id/menu_wish_list")
        wishlist_icon.click()
        courses_in_wishlist =
driver.find_elements_by_id("com.udemy.android:id/browse_course_card")
        assert len(courses_in_wishlist) > 0
    def test_remove_course_from_wishlist(driver):
        wishlist_icon =
driver.find_element_by_id("com.udemy.android:id/menu_wish_list")
        wishlist_icon.click()
```

Продовження додатка В

```
        course=driver.find_element_by_id("com.udemy.android:id/course_thumbnail_im
age")
            course.click()
            wishlisted_button                                                    =
driver.find_element_by_id("com.udemy.android:id/course_wishlisted_button")
            wishlisted_button.click()
            driver.back()
            courses_in_wishlist                                                  =
driver.find_elements_by_id("com.udemy.android:id/browse_course_card")
            assert len(courses_in_wishlist) == 0


        #Fifth test-case
        def test_add_course_to_cart_from_wishlist(driver):
            sleep(10)
            wishlist_icon                                                        =
driver.find_element_by_id("com.udemy.android:id/my_wishlist_item")
            wishlist_icon.click()
            course = driver.find_element_by_id("com.udemy.android:id/card_view")
            course.click()
            add_to_cart_button                                                   =
driver.find_element_by_id("com.udemy.android:id/add_to_cart_button")
            add_to_cart_button.click()
            cancel_button                                                        =
driver.find_element_by_id("com.udemy.android:id/btn_cancel")
            cancel_button.click()
            add_to_cart_button.click()
            alert_dialog = driver.find_element_by_id("com.udemy.android:id/alertTitle")
```

```
        assert alert_dialog.text == "Adding this course to your cart will remove it from your wishlist"
        ok_button = driver.find_element_by_id("android:id/button1")
        ok_button.click()
        recommended_courses = driver.find_element_by_id("com.udemy.android:id/title")
        assert recommended_courses.text == "Recommended for You"
        page_layout = driver.find_element_by_id("com.udemy.android:id/detail_container")
        page_layout.click()
        go_to_cart_button = driver.find_element_by_id("com.udemy.android:id/go_to_cart_button")
        assert go_to_cart_button.text == "Go to cart"
        cart_icon = driver.find_element_by_id("com.udemy.android:id/my_cart_item")
        cart_count = driver.find_element_by_id("com.udemy.android:id/my_cart_item_badge")
        assert cart_count.text == "1"


    #Sixth test-case
    def test_udemy_app(driver):
        sleep(5)
        search_icon = driver.find_element_by_id("com.udemy.android:id/search_icon")
        search_icon.click()
        sleep(2)
        browse_category = driver.find_element_by_xpath("//android.widget.TextView[@text='Development']")
        browse_category.click()
```

```
        sleep(2)
        first_course = driver.find_element_by_id("com.udemy.android:id/course_card_view")
        first_course.click()
        sleep(2)
        add_to_wishlist = driver.find_element_by_id("com.udemy.android:id/wishlist_button")
        add_to_wishlist.click()
        sleep(2)
        back_button = driver.find_element_by_xpath("//android.widget.ImageButton[@content-desc='Navigate up']")
        back_button.click()
        sleep(2)
        second_course = driver.find_elements_by_id("com.udemy.android:id/course_card_view")[1]
        second_course.click()
        sleep(2)
        add_to_cart = driver.find_element_by_id("com.udemy.android:id/add_to_cart_button")
        add_to_cart.click()
        sleep(2)
        continue_to_cart = driver.find_element_by_id("com.udemy.android:id/continue_to_cart_button")
        continue_to_cart.click()
        sleep(2)
        assert "Cart" in driver.page_source, "Cart page not opened"
```

Продовження додатка В

```
        wishlist_section = driver.find_element_by_id("com.udemy.android:id/wishlist_card_view")
        wishlist_section.click()
        sleep(2)
        remove_button = driver.find_element_by_id("com.udemy.android:id/remove_course_button")
        remove_button.click()
        sleep(2)
        assert "Removed" in driver.page_source, "Course removed from Wishlist section"
        cart_item_section = driver.find_elements_by_id("com.udemy.android:id/cart_item")[1]
        save_for_later_button = cart_item_section.find_element_by_id("com.udemy.android:id/save_for_later_button")
        save_for_later_button.click()
    sleep(2)
    assert "Saved" in driver.page_source, "Course saved for later"
    saved_courses_section = driver.find_element_by_id("com.udemy.android:id/saved_courses_card_view")
    saved_courses_section.click()
    sleep(2)
    remove_button = driver.find_element_by_id("com.udemy.android:id/remove_course_button")
    remove_button.click()
    sleep(2)
    assert "Removed" in driver.page_source, "Course removed from saved courses section"
```

Продовження додатка В

```
    #Seventh test-case
    def test_udemy_cart(driver):
        sleep(5)
        search_icon = driver.find_element_by_id("com.udemy.android:id/search_icon")
        search_icon.click()
        sleep(2)
        top_search = driver.find_element_by_xpath("//android.widget.TextView[@text='python']")
        top_search.click()
        sleep(2)
        course_1 = driver.find_elements_by_id("com.udemy.android:id/course_card_view")[0]
        course_1.click()
        sleep(2)
        add_to_cart_1 = driver.find_element_by_id("com.udemy.android:id/add_to_cart_button")
        add_to_cart_1.click()
        sleep(2)
        carousel_course_2 = driver.find_elements_by_id("com.udemy.android:id/carousel_course_card_view")[0]
        carousel_course_2.click()
        sleep(2)
        add_to_cart_2 = driver.find_element_by_id("com.udemy.android:id/add_to_cart_button")
        add_to_cart_2.click()
        sleep(2)
```

```
        carousel_course_3 =
driver.find_elements_by_id("com.udemy.android:id/carousel_course_card_view")[1]
        carousel_course_3.click()
        sleep(2)
        add_to_cart_3 =
driver.find_element_by_id("com.udemy.android:id/add_to_cart_button")
        add_to_cart_3.click()
        sleep(2)
        cart_icon = driver.find_element_by_id("com.udemy.android:id/cart_icon")
        cart_icon.click()
        sleep(2)
        remove_button_1 =
driver.find_elements_by_id("com.udemy.android:id/remove_course_button")[0]
        remove_button_1.click()
        sleep(2)
        cancel_button = driver.find_element_by_id("android:id/button


    #Eighth test-case
    def test_add_to_wishlist_cart(driver):
        search_icon = driver.find_element_by_id("com.udemy.android:id/search_icon")
        search_icon.click()
        search_keyword =
driver.find_element_by_id("com.udemy.android:id/autosuggest_search_text")
        search_keyword.send_keys("python\n")
        course_1 =
driver.find_element_by_xpath("//android.widget.TextView[@text='Course 1']")
        course_1.click()
```

Продовження додатка В

```
        add_to_cart_button =
driver.find_element_by_id("com.udemy.android:id/course_buy_button")
        add_to_cart_button.click()
        continue_to_cart_button =
driver.find_element_by_id("com.udemy.android:id/continue_button")
        continue_to_cart_button.click()
        cart_course =
driver.find_element_by_id("com.udemy.android:id/cart_list_item")
        cart_course.click()
        add_to_wishlist_button =
driver.find_element_by_id("com.udemy.android:id/course_add_to_wishlist_button")
        add_to_wishlist_button.click()
        popup_text =
driver.find_element_by_id("com.udemy.android:id/dialog_warning_text")
        assert popup_text.text == "Adding this course to your wishlist will remove it
from your cart"
        ok_button =
driver.find_element_by_id("com.udemy.android:id/dialog_confirm_button")
        assert ok_button.text == "Ok"
        ok_button.click()
cart_icon=driver.find_element_by_id("com.udemy.android:id/cart_menu_item")
        cart_icon.click()
        empty_cart_text =
driver.find_element_by_id("com.udemy.android:id/cart_empty_text")
        assert empty_cart_text.text == "Your cart is empty."
```

Продовження додатка В

```
    #Ninth test-case
    def test_signin_course(driver):
        account_icon = driver.find_element_by_accessibility_id("Account")
        account_icon.click()
        sleep(3)
        sign_out_btn =
driver.find_element_by_id("com.udemy.android:id/sign_out_button")
        sign_out_btn.click()
        sleep(2)
        sign_out_pop_up =
driver.find_element_by_id("com.udemy.android:id/sign_out_pop_up_layout")
        assert sign_out_pop_up.is_displayed()
        sign_out =
driver.find_element_by_id("com.udemy.android:id/sign_out_button_popup")
        sign_out.click()
        sleep(5)
        browse_btn = driver.find_element_by_accessibility_id("Browse")
        browse_btn.click()
        sleep(5)
search_icon=driver.find_element_by_id("com.udemy.android:id/search_src_text")
        search_icon.click()
        sleep(2)
search_field=driver.find_element_by_id("com.udemy.android:id/search_src_text")
        search_field.send_keys("python")
        sleep(3)
```

Продовження додатка В

```
        course = driver.find_element_by_xpath("//android.widget.TextView[contains(@text,'Python for Beginners')]")
        course.click()
        sleep(5)
        add_to_cart_btn = driver.find_element_by_id("com.udemy.android:id/add_to_cart_button")
        add_to_cart_btn.click()
        sleep(2)
sign_in_btn=driver.find_element_by_id("com.udemy.android:id/sign_in_button_cart")
        sign_in_btn.click()
        sleep(2)
        sign_in_pop_up = driver.find_element_by_id("com.udemy.android:id/sign_in_pop_up_layout")
        assert sign_in_pop_up.is_displayed()
        sign_in_email = driver.find_element_by_id("com.udemy.android:id/email")
        sign_in_email.send_keys("auto.testing541@gmail.com")
        sleep(2)
sign_in_password=driver.find_element_by_id("com.udemy.android:id/password")
        sign_in_password.send_keys("1111")
        sleep(2)
        sign_in_btn = driver.find_element_by_id("com.udemy.android:id/sign_in_button")
        sign_in_btn.click()
        sleep(2)
        error_message = driver.find_element_by_id("com.udemy.android:id/password_input_layout")
```

Продовження додатка В

```
        password_field =
driver.find_element_by_id("com.udemy.android:id/password_input")
        password_field.send_keys("testTEST")
        driver.hide_keyboard()
        login_btn = driver.find_element_by_id("com.udemy.android:id/submit_button")
        login_btn.click()
        sleep(2)
        account_icon =
driver.find_element_by_id("com.udemy.android:id/account_icon")
        assert account_icon.is_displayed()
        driver.quit()
```