

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

«До захисту допущено»

Завідувач кафедри

_____ Віталій Романкевич
(підпис)

“___” червня 2021 р.

**Дипломний проєкт
на здобуття ступеня бакалавра**

зі спеціальності **123 «Комп'ютерна інженерія»**

на тему: Комп'ютерні засоби для автоматизації тестування додатків

Виконав: студент IV курсу, групи КВ-74
Сенеджук Андрій Юрійович

(підпис)

Керівник, доц. каф. СП і СКС, к. т. н. Тесленко О. К.

(підпис)

Консультант з нормоконтролю, доц.каф.СПСКС, к.т.н. Клятченко Я.М.

(підпис)

Рецензент _____

(підпис)

Засвідчую, що у цьому дипломному проєкті
немає запозичень з праць інших авторів без
відповідних посилань.

Студент _____
(підпис)

Київ – 2021 року

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 123 «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Віталій Романкевич
(підпис)
«__» червня 2021 р.

**ЗАВДАННЯ
на дипломний проєкт студента
Сенеджука Андрія Юрійовича**

1. Тема проєкту «Комп'ютерні засоби для автоматизації тестування додатків», керівник проєкту Тесленко О. К., кандидат технічних наук, доцент кафедри системного програмування і спеціалізованих комп'ютерних систем, затверджені наказом по університету від «__» _____ 2021 р. № _____
2. Термін подання студентом проєкту
3. Вихідні дані до проєкту: див. Технічне завдання
4. Зміст пояснювальної записки
 - Автоматизація тестування додатків
 - Тестування «Розробки через поведінку» трафіку додатків
 - Опис технологія тестування
 - Реалізація комп'ютерних засобів для автоматизації тестування додатків
 - Тестування аналізатора
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень, плакатів, презентацій тощо)

- Принцип роботи розроблених комп'ютерних засобів для реалізації тестування додатку. Схема роботи.
- Схема отримання звіту щодо виконання автотестів.
- Принцип поповнення чеклисту тестовими сценаріями. Схема роботи.
- Схема заповнення автотестами набору тестів регресу.
- Презентація за темою роботи.

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
нормоконтроль	Клятченко Я. М., к. т. н., доцент каф. СП і СКС,		

7. Дата видачі завдання 30 жовтня 2021 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проєкту	15.03.2021	
2.	Розроблення та узгодження технічного завдання	30.03.2021	
3.	Аналіз існуючих рішень	05.04.2021	
4.	Підготовка матеріалів ПЕРШОГО розділу дипломного проєкту	15.04.2021	
5.	Підготовка матеріалів ДРУГОГО розділу дипломного проєкту	20.04.2021	
6.	Підготовка матеріалів третього розділу дипломного проєкту	26.04.2021	
7.	Підготовка матеріалів четвертого розділу дипломного проєкту	05.05.2021	
8.	Розробка комп'ютерних засобів	10.05.2021	
9.	Підготовка графічної частини дипломного проєкту	18.05.2021	
10.	Оформлення документації дипломного проєкту	23.05.2021	
11.	Попередній огляд матеріалів на кафедрі	26.05.2021	

Студент

(підпис)

Андрій СЕНЕДЖУК

Керівник проєкту

Олександр ТЕСЛЕНКО

АНОТАЦІЯ

Кваліфікаційна робота включає пояснювальну записку (55 с., 26 рис.).

Об'єкт розробки – створення комп'ютерних засобів для автоматизації тестування додатків.

Комп'ютерні засоби дозволяють: запровадити автоматизацію тестування додатків для мобільних застосунків на базах iOS та Android; перевірити правильність роботи додатків у короткий проміжок часу; провести перевірку логіки роботи програми і правильність роботи API сервера.

В ході розробки:

- проведено аналіз методів побудови автоматизованого тестування для додатків;
- сформульовані вимоги до комп'ютерних засобів автоматизації тестування;
- розроблена система автоматизації управління автоматизованим тестуванням;
- розроблено тести на мові Gherkin, зав'язані на принципах BDD;
- розроблено модуль інтеграції клієнт - REST API та виконана реалізація тестів мовою програмування Ruby;
- виконана автоматизація запуску компонентів тестування;
- розроблено правила перехоплення трафіку від тестового додатку до серверу, перенаправлення запитів до мок-серверу за допомогою mitm технології;

Упровадження цієї системи засобів дозволить уникнути комунікаційних проблем між замовником та командою розробників та забезпечити якісне тестування програми в умовах, коли вимоги замовника можуть змінюватись.

Ключові слова:

BDD, REST API, RUBY, MOCK-SERVER, MITM, GRENKIN, IOS, ANDROID, ТЕСТУВАННЯ, ТЕСИ, АВТОМАТИЗАЦІЯ

ABSTRACT

Qualification work includes an explanatory note (55 pages, 26 figures).

The object of development is the creation of computer tools to automate application testing.

Computer tools allow you to: implement automation of testing applications for mobile applications based on iOS and Android; check the correct operation of applications in a short period of time; to check the logic of the program and the correct operation of the ARI server.

During development:

- analysis of methods for building automated testing for applications;
- formulated requirements for computer testing automation tools;
- the system of automation of management of the automated testing is developed;
- developed Gherkin tests based on BDD principles;
- the client integration module - REST API was developed and tests were implemented in the Ruby programming language;
- automation of testing components startup is performed;
- developed rules for intercepting traffic from the test application to the server, redirecting requests to the mock server using mitm technology;
- developed a mock server, which provides the necessary responses to application requests.

The implementation of this system of tools will avoid communication problems between the customer and the development team and ensure quality testing of the program in conditions where customer requirements may change.

Keywords:

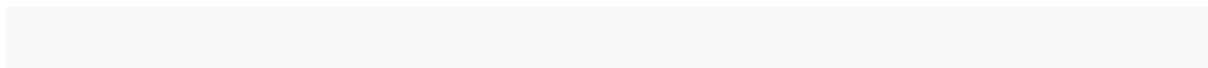
BDD, REST API, RUBY, MOCK-SERVER, MITM, GRENKIN, IOS, ANDROID, TESTING, TESTS, AUTOMATION

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки	
	A4	ІАЛЦ.045490.002 ТЗ	Комп'ютерні засоби для автоматизації тестування додатків. Технічне завдання	4			
	A4	ІАЛЦ.045490.003 ТП	Комп'ютерні засоби для автоматизації тестування додатків. Відомість технічного проекту.	1			
	A4	ІАЛЦ.045490.004 ПЗ	Комп'ютерні засоби для автоматизації тестування додатків. Пояснювальна записка	55			
	A4	ІАЛЦ.045490.005 Д1	Принцип розроблених комп'ютерних засобів для реалізації автоматизованого додатку. Схема роботи.	1			
ІАЛЦ.045490.003 ТП							
Змін.	Арк.	№ докум.	Підпис	Дата			
Розробив	Сенеджук А. Ю.						
Перевірив	Тесленко О. К.						
Консульт.							
Н. контроль	Клятченко Я.М.						
Зав. каф.	Романкевич В.О.						
Комп'ютерні засоби для автоматизації тестування додатків					Літ.	Аркуш	Аркушів
						1	2
Відомість технічного проекту					КПП ім. Ігоря Сікорського, ФПМ KB-74		

[illegible]

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ.....	2
2. ПІДСТАВА ДЛЯ РОЗРОБКИ	2
3. ЦІЛЬ І ПРИЗНАЧЕННЯ РОБОТИ	2
4. ДЖЕРЕЛА РОЗРОБКИ.....	2
5. ТЕХНІЧНІ ВИМОГИ.....	2
5.1. Вимоги до програмного продукту, що розробляється	2
5.2. Вимоги до апаратного забезпечення.....	3
5.3. Вимоги до програмного та апаратного забезпечення користувача	3
6. ЕТАПИ РОЗРОБКИ	4



					ІАЛЦ. 045490.002 ТЗ			
Зм	Лист	№ докум.	Підп.	Дата	Комп'ютерні засоби для автоматизації тестування додатків Технічне завдання	Лім.	Лист	Листів
Розроб.	Сенеджук А Ю							
Перев.	Тесленко О. К.						1	4
Н. контр.	Клятченко Я. М.					КПІ ім. Ігоря Сікорського, ФПМ, КВ-74		
Затв.	Романкевич В.О.							

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ

Назва розробки: «Комп'ютерні засоби для автоматизації тестування додатків».

Галузь застосування: організація процесу автоматизованого тестування мобільних додатків у IT-компаніях.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на дипломне проектування на здобуття першого (бакалаврського) рівня вищої освіти, затверджене кафедрою системного програмування і спеціалізованих комп'ютерних систем Національного технічного університету України «Київський Політехнічний Інститут імені Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОБОТИ

Метою даного проекту є створення комп'ютерних засобів для автоматизації тестування додатків.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом інформації є технічна та науково-технічна література, технічна документація, публікації у періодичних виданнях та електронні статті у мережі Інтернет.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до програмного продукту, що розробляється

- сумісність з будь якою операційною системою (IOS, Android);
- можливість управління процесом обробки тестових сценаріїв;
- можливість додавати нові тестові сценарії;

					ІАЛЦ.045490.002 ТЗ	Лист 2
Зм	Лист	№ докум.	Підп.	Дата		

- можливість редагування існуючих тестових сценаріїв;
- наявність системи контакту з виводом результатів;
- прив'язки файлів з тестовими сценаріями до результатів;
- зберігання тестових сценаріїв;
- аналітичні можливості.

5.2. Вимоги до апаратного забезпечення

- Процесор: 2,4-ядерний, MediaTek, Snapdragon, Kirin, Intel Atom;
- Оперативна пам'ять: 2 Гб;
- Наявність доступу до мережі Internet (Ethernet, GPRS, EDGE, 3G, 4G);

5.3. Вимоги до програмного та апаратного забезпечення користувача

- Операційна система Windows iPhone 8, Android, iOS;

					ІАЛЦ.045490.002 ТЗ	Лист
						3
Зм	Лист	№ докум.	Підп.	Дата		

6. ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів
1.	Постановка задачі керівником відділу. Вивчення літератури по темі.	12.04
2.	Пошук та аналіз існуючих рішень.	17.04
3.	Розробка архітектури відповідно до поставленої задачі.	22.04
4.	Програмування модулів додатку. Вивчення матеріалів проекту.	27.04
5.	Тестування розробленого програмного коду.	03.05
6.	Виправлення помилок та невідповідностей у програмному коді.	08.05
7.	Робота з відділом тестування, внесення правок, реалізація додаткового функціоналу.	12.05
8.	Демонстрація готового проекту керівникові відділу.	16.05

ЗМІСТ

Перелік скорочень, умовних позначень, термінів	2
ВСТУП	4
1. АВТОМАТИЗАЦІЯ ТЕСТУВАННЯ МОБІЛЬНИХ ЗАСТОСУНКІВ.....	6
1.1. Поняття автоматизованого тестування і його значення для розробки програмного забезпечення	6
Тестування є дуже важливою та невід’ємною частиною процесу розробки програмного забезпечення та входить у число найбільш ефективних та потрібних способів забезпечення якості продуктів, при чому під якістю мається на увазі зручність користування, а не тільки надійність програми. ____	6
1.2. Рівні автоматизації. Місце тестування користувацького інтерфейсу в загальному процесі автоматизованого тестування	9
1.3. Інструменти для автоматизації тестування користувацького інтерфейса	11
1.4. Методи тестування мобільних застосунків	13
1.5. Існуючі способи автоматизації. Підходи. Паттерни.	14
2. Тестування «розробки через поведінку» трафіку мобільних застосунків .	19
2.1. Технологія «Розробки через поведінку»	19
2.2 Proxu	20
2.3 Криптографічна атака MitM	22
2.3 Інструменти для реалізації автоматизованого тестування	24
2.3.1 Емулятор для генерації поведінки віртуального мобільного пристрою - AVD	24
2.3.2 Genymotion	25
2.3.3 Bluetacks	26
2.3.4 MITMf	27

					ІАЛЦ.045440.004 ПЗ						
Зм.	Арк.	№ докум.	Підп.	Дата	«Комп’ютерні засоби для автоматизації тестування додатків». Пояснювальна записка	Літ.		Аркуш		Аркушів	
Розроб.		Сенеджук А. Ю.						1			
Перевір.		Тесленко О. К.									
Н. контр.		Клятченко Я.М.									
Затв.		Романкевич В. О				КПІ ім. Ігоря Сікорського, ФПМ КВ-74					

2.3.5 DrodSheep	27
2.3.6 MitmProxy	28
3. ОПИС ТЕХНОЛОГІЇ ТЕСТУВАННЯ	29
3.1 Опис додатку для автоматизованого тестування	29
3.2 Cucumber	30
3.3 Appium	33
3.4 Mitmproxy	33
3.5 Mock-server	34
3.6 Опис структури API для взаємодії з мобільним додатком	35
3.6.1 Взаємодія із API та її методи	35
3.7 Схема роботи	40
4. РЕАЛІЗАЦІЯ КОМП'ЮТЕРНИХ ЗАСОБІВ ДЛЯ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ ДОДАТКІВ	42
4.1 Створення тестів мовою Gherkin	42
4.2 Розробка Steps to reproduce	44
4.3 Перевірка та запуск комп'ютерних засобів	45
4.4 Запуск Мок-серверу. Симуляція роботи API-серверу	46
4.4.1 Перевірка послідовності запитів	47
4.5 Виконання роботи комп'ютерних засобів. Перехоплення трафіку	48
4.6 Застосування розроблених комп'ютерних засобів	49
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ	54

Перелік скорочень, умовних позначень, термінів

Бізнес-аналітик – особа, яка займається створенням документації, згідно з якою відбувається розробка програмного продукту

ПЗ – програмне забезпечення

Приймальне тестування – це тестування ПЗ яке проводять для перевірки на відповідність до вимог замовника. Приймальне тестування — це фінальний етап тестування програми перед публічним запуском.

Регресивне тестування – це тестування, яке задіює перевірку абсолютно усього функціоналу програмного продукту.

Скрипт – написаний на мові програмування код, який виконує певні задачі, які задаються програмістом/тестувальником.

Тестувальник – особа, яка здійснює перевірку правильності роботи програмного проєкту та аналізує готовність продукту

Android – програмний код, що зберігається у децентралізованій, розподіленій мережі Blockchain, а також визначає умови, з якими згодні всі сторони, що використовують контракт.

iOS – група алгоритмів, що переводять інформацію, що знаходиться за межами мережі, у зрозумілий блокчейну формат

SSL – протокол для веббравзерів, який дозволяє проводити автентифікацію, шифрування та дешифрування даних.

ISO/IEC12207 –стандарт ISO, що описує процеси життєвого циклу програмного забезпечення.

URL – Uniform Resource Locator.

					ІАЛЦ.045440.004 ПЗ	Лист 3
Зм	Лист	№ докум.	Підп.	Дата		

ВСТУП

Сучасний світ не можливо уявити без використання програмного забезпечення. Важко знайти бодай одну галузь, де не використовують код. Величезні кошти витрачаються щодня на розробку різноманітних програм, проєктів, які такі необхідні у бізнесі, науці, медицині тощо. Тому першостепенною задачею стоїть зниження витрат на розробку програм та покращення якості продукту.

Перед компаніями, які займаються розробкою програмного забезпечення для замовників, стоїть одна із важливих задач: перед оголошенням готовності продукту, його потрібно протестувати. Тестування є невід'ємною складовою фундаментальної структури, яка визначає задачі до виконання на кожному етапі процесу розробки програмного забезпечення. ISO/IEC 12207 є міжнародним стандартом для процесів життєвого циклу розробки програмного продукту.

Тестування – це процес пошуку невідповідностей актуального стану програмного забезпечення до очікуваного, за допомогою певного набору тестових сценаріїв, які вибираються для певних умов. Головною ідеєю є пошук невідповідностей для запобігання виявлень дефектів кінцевими користувачами та знизити ціну помилки у програмному продукті, знайшовши її на якомога ранішому етапі розробки.

Автоматизація тестування дозволяє значно скоротити витрати компаній-розробників, затрати на тестування, зекономити ресурси та час, знизити ризик випуску дефектного програмного забезпечення. Ці важливі питання є головними для бізнес ідей, тому технології автоматизованого тестування набирають все більшої і більшої популярності серед компаній-розробників, чи певних зацікавлених лиць, адже економія ресурсів лежить на першому місці. Зібравши в купу основні проблеми, які вирішує автоматизація тестування, визначилась й актуальність теми, яка вибрана для написання дипломного проєкту.

					ІАЛЦ.045440.004 ПЗ	Лист 4
Зм	Лист	№ докум.	Підп.	Дата		

Мобільні застосунки на сучасному ринку – найбільш перспективна ідея для ведення бізнесу. Хоча тестування програмних продуктів існує уже давно, саме автоматизація тестування мобільних застосунків являється відносно молодим родом діяльності. Технології, інструменти та ресурси, які використовуються у цій певній області, відносно мало вивчені і тільки виходять на ринок та впевнено завойовують прихильність та довіру компаній-розробників. Не зважаючи на це, автоматизована структура тестування для мобільних застосунків дуже необхідна, так як на сьогоднішній день лінійка моделей мобільних телефонів, оболочок, операційних систем, розмір діагоналей екрану, портретні та альбомні режими, різні з'єднання з інтернетом, вай-фаєм, енергозберігаючі режими тощо дуже різноманітна і потребує тестування. Це визначає практичну новизну дипломного проєкту.

Предмет дослідження дипломного проєкту є розробка комплексу програм для автоматизації тестування мобільних застосунків.

Ціллю проєкту є автоматизація процесу тестування застосунку, розробка тестів для тестування користувацького інтерфейса під операційні системи iOS та Android, тест продуктивності застосунку та регресивне та функціональне тестування.

					ІАЛЦ.045440.004 ПЗ	Лист
						5
Зм	Лист	№ докум.	Підп.	Дата		

1. АВТОМАТИЗАЦІЯ ТЕСТУВАННЯ МОБІЛЬНИХ ЗАСТОСУНКІВ

1.1. Поняття автоматизованого тестування і його значення для розробки програмного забезпечення

Тестування є дуже важливою та невід'ємною частиною процесу розробки програмного забезпечення та входить у число найбільш ефективних та потрібних способів забезпечення якості продуктів, при чому під якістю мається на увазі зручність користування, а не тільки надійність програми.

Під якістю програмного забезпечення розуміють сукупність його властивостей, які обумовлені його придатністю задовільняти задані і висвітлені потреби у відповідності з його призначенням, ГОСТ 28806-90 [16]. Таким чином, якісним рахується той продукт, який відповідає критеріям якості, пред'явлені зацікавленими особами, компаніями-замовниками або ж звичайними користувачами. Програмне забезпечення повинно відповідати визначеним стандартам і очікуванням для того, щоб його можна було вважати якісним та відповідним.

Згідно зі словами Єдсгера Вібе Дейкстри, тестування програми може доволі ефективно продемонструвати наявність помилок, але безнадійно неадекватно для демонстрації їх відсутності [17]. І знову ж таки, під помилками чи дефектами програмного продукту розуміють невідповідності у роботі програмного продукту, робота проєкту, не описану у документації компанії-замовника. Тобто по суті, основна задача процесу тестування є виявлення розбіжностей між реальною поведінкою системи та очікуваної, яка відповідає поставленим до програмного продукту вимогам зі сторони компанії-замовника, чи зацікавлених осіб.

На практиці тестування здійснюється шляхом виконання певних тестових сценаріїв, які написані відповідальними особами на основі документації, розробленої бізнес-аналітиками відповідно до вимог зі сторони компанії-замовника програмного продукту. Виконується цей процес як в ручну, так і автоматично. Під ручним тестуванням розуміють виконання

					ІАЛЦ.045440.004 ПЗ	Лист 6
Зм	Лист	№ докум.	Підп.	Дата		

певних наборів тестових сценаріїв(дій у тестовому артефакті) певними особами без використання сторонніх програм. Звіряється очікуваний результат із реальним та перевіряються основні функції мобільного застосунку. Автоматичне тестування виконується шляхом використання певних інструментів і називається автоматизованим тестуванням програмного забезпечення.

У автоматизованого тестування є як переваги, так і недостатки. Перевагами можна назвати:

- Повторюваність – написані тести за допомогою певних артефактів виконуються одноманітно, що виключає похибку людського фактора
- Швидке виконання – автоматизований скрипт не звіряється із документацією, що значно скорочує час
- Відсутність впливу «людського фактору»
- Виконання тест-кейсів особливої складності, які недоступні та не підвласні людині
- Властивість зберігати, аналізувати у зручній формі величезного об'єму даних
- Властивість виконувати низькорівневі дії з застосунком, операційною системою, проєктом

До недостаткам автоматизованого тестування відносяться:

- Необхідність зв'язку з висококваліфікованим персоналом для автоматизації, натомість можливість використовувати низькорівневу кваліфіковану працю реальних тестувальників
- Витрати на засоби для організації автоматизованого тестування, на розробку та підтримку тестів, їх написання
- Грошові витрати та ризики, які пов'язані із ймовірністю мати великий вибір засобів автоматизації та складність вибору
- Можливе застаріння рішень, які були використані при реалізації написання авто-тестів, зміни вимог, зміни самого інтерфейса проєкта

					ІАЛЦ.045440.004 ПЗ	Лист
Зм	Лист	№ докум.	Підп.	Дата		7

Автоматизація тестування не дозволяє повністю обійтися без задіявання мануального(ручного) тестування, але значно допомагає прибрати рутинні дії із процесу тестування програмного продукту, також, понизити вартість програми та значно покращити якість та пришвидчити терміни виконання якісного програмного проєкту.

Тестування є одним із найбільш громістких етапів розробки програмного забезпечення, які займають велику кількість часу, при чому ціна виправлення помилки, напряду залежить від часу, яке проходить з моменту її виникнення в залежності до моменту виявлення дефекту.

У своїй книгу «A Practitioner`s Guide to Software Test Design» Лі Конпланд приводив відповіді, які дають спеціалісти по тестуванню на питання про те, з якими найбільш розповсюдженими проблемами приходить зійтнутись при розробці програмного проєкту: «недостатньо часу, щоб тестувати ретельно», «надто багато комбінацій вхідних даних», «недостатньо часу, щоб протестувати вдало», «надто мало часу виділяється на тестування»[3].

Майже усе, що перелічено вище, пов'язані з об'ємом роботи, необхідна до виконання, і кількість часу, яке необхідно буде витратити для того, щоб якісно протестувати програмну розробку. Згідно з цим ж, ідея про автоматизацію процесу тестування, яке й дозволяє вирішити питання з нехваткою часу для проведення успішного та якісного тестування, стає якомога актуальнішим по мірі збільшення складності програмних продуктів, що розробляються, і ріст вимог відповідно до їх якості.

Перші спроби запровадити автоматизації у процес тестування мали місце ще у 80-х роках. Близько 90-го року з'явилися перші інструменти та засоби автоматизації тестування програмного забезпечення. А уже ближче до двохтисячних років автоматизації тестування уже повністю стала невід'ємною частиною процесу створення програмного проєкту, як такого.

Сьогодні ж не можливо уявити собі розробку гігантського програмного продукту без використання процесів автоматизації на кожному з етапів розробки, особливо на стадії тестування. Сучасний етап розвитку тестування характеризується глибокою інтеграцією тестування з процесом розробки в цілому, а також величезним використанням автоматизації, просто колосальним набором технології та інструментальних засобів для автоматизації [19].

1.2. Рівні автоматизації. Місце тестування користувацького інтерфейсу в загальному процесі автоматизованого тестування

Сьогодні є безліч різних підходів, які слугують для автоматизованого тестування. Комплексна та загальна стратегія тестування включає в себе три основних рівня автоматизації:

1. Рівень модульного тестування(Unit Tests) – тести, які стосуються компонент(певних функцій) програмного проєкту, Зазвичай ці тести пишуться програмістами, розробниками продукту.

2. Рівень функціонального тестування(Functional Tests or Service/API Tests) – тестування за допомогою доступу до функціонального шару продукту, без використання та задіяння інтерфейса, створеного для кінцевого користувача.

3. Рівень тестування користувацького інтерфейсу(GUI Tests) – тестування не тільки самого інтерфейсу, але й виконання операцій, яка використовує бізнес логіку застосунку, реалізація функціонального тестування.

Для забезпечення якомога кращої якості застосунку рекомендується автоматизувати всі три рівні, перераховані вище. На рисунку 1 приведено класичну піраміду, яка слугує для побудови автоматизованого тестування при розробці програмного застосунку [18].

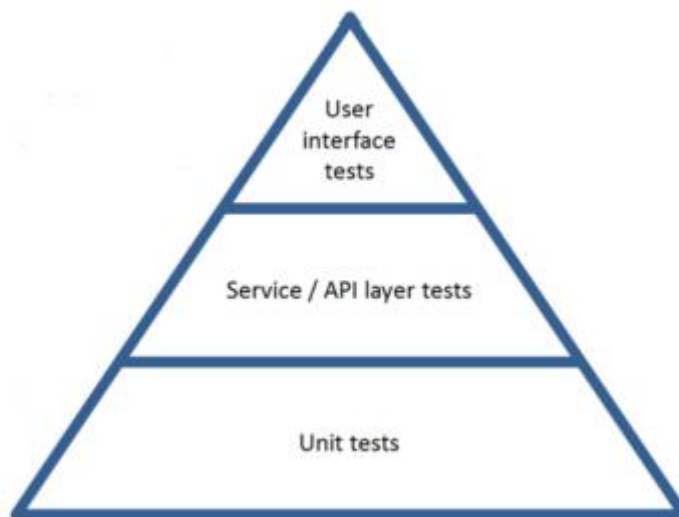


Рисунок 1.2 – Класична піраміда автоматизації

Даний дипломний проєкт описуватиме реалізацію третього рівня тестування – тестування користувацького інтерфейса. Автоматизація тестування програмних продуктів через графічний користувацький інтерфейс набирає все більшої популярності. Врахувавши, якщо застосунок буде тестуватись тим ж способом, що й при використанні кінцевим користувачем.

Плюсами такого виду тестування є те, що тестування проходить на реальних пристроях, але з різними конфігураційними файлами, з різними операційними системами, оболочками, браузерами тощо. Такий вид тестування є найбільш важливим видом тестування мобільних застосунків, оскільки необхідно здійснювати тестування на величезній лінійці телефонів сьогодення, яка нараховує більше тисячі різних телефонів від різних компаній-виробників. А якщо враховувати кількість версій операційних систем на телефонах, то стає очевидним той факт, що абсолютно все протестувати тестувальникам стає нереальною задачею, що не скажеш про автоматизацію. Всього пару годин виконання автотестів і вся лінійка, яка буде заявлена у офіційних документах про підтримку і буде результат.

Мінусами такого виду тестування можна назвати те, що тестування проходять довго, багато сил витрачається на їх підтримку (при зміні інтерфейса потребується внесення змін у всі написані тести, які

використовують оновлені елементи), тести можуть бути нестабільними (є великі шанси, що тести можуть «падати» через спроби звернутись до елементів інтерфейса, які ще не підгрузились, нестабільно працюють з певними конфігураціями).

Реалізується такий вид автоматизованого тестування через використання спеціальних інструментів та утиліт, які надають можливість взаємодії з інтерфейсом програмного продукту. При компіляції у застосунку додається додаткова бібліотека, а у код додаються методи для виклику даної бібліотеки. Після цього розробник тестів отримує доступ до елементів інтерфейсу та їх налаштувань, міняє цю інформацію та імітує різні дії користувача, які виконувались б при реальному користуванню застосунків кінцевими користувачами: натиски по екрані, здійснення скролу тощо.

1.3. Інструменти для автоматизації тестування користувацького інтерфейса

Щоб реалізувати автоматизації тестування при розробці програмного продукту, слід врахувати, що автоматизація потребує інструменти для її реалізації. В якості інструментів можна використовувати комерційні, безкоштовні, або ж власно розроблені продукти для тестування.

У кожного з вищеперерахованих видів є свої плюси й мінуси.

До плюсів використання комерційних продуктів можна віднести:

- Величезна економія часу, який витрачається на розробку автотестів, не потрібно писати самостійно
- Чудове представлення з готовими інструментами, що дає можливість зручно писати і проводити відладку самостійно
- Підтримка зі сторони розробників, компанія яких і займалась цим продуктом. Є можливість звернутись та домовитись про реалізацію додаткового інструменту, не витрачаючи свого часу на розробку.

До мінусів використання комерційних продуктів можна віднести:

- Висока вартість софту

- Відсутність доступу до вихідного коду програми
- Відсутність гнучкості у виборі підходу тестування, строге наслідування моделі, база якої і лежить у інструменті

До плюсів використання безкоштовних інструментів відноситься:

- Помірна вартість або взагалі її відсутність
- Доступність до вихідного коду проєкту

До мінусів використання безкоштовних інструментів відноситься:

- Недостатня функціональність
- Ризик відмови у підтримці програмного інструменту

До плюсів використання власних інструментів автоматизації можна віднести:

- Можливість розробки інструмента як побічного результату процесу створення самого застосунку

- Доступність вихідного коду
- Гнучкість у розробці утиліти
- Можливість приймати зміни уже у процесі розробки та після неї

До мінусів використання власних інструментів автоматизації можна віднести:

- Відсутність гарантів у стабільній роботі
- Ризики, які пов'язані з тим, що функціонал інструменту може виявитись недостатньої якості для виходу проєкту на ринок

Найбільш відомити комерційними інструментами для автоматизації тестування користувацького інтерфейса мобільних застосунків є: TestComplete, Ranorex, SeeTest Automation, Squish. Стосовно безкоштовних інструментів: Appium, Selenium, Selendroid, Robotium, Calabash.

Для вибору та оцінки інструментів слід враховувати наступні критерії [21]:

- Підтримка мобільних платформ (упізнання необхідне елементам інтерфейса, технології взаємодії з ними, як реалізований процес взаємодії)
- Підтримка типів застосунків (веб, гібрид, нативні)
- Підтримка запуску тестів на емуляторах, реальних пристроях
- IDE (підтримка мов програмування, зручність використання, інструменти дебагу)
- Менеджери тестів (можливість по конфігурації тестових виборів, можливість паралельного запуску тестів, налаштування параметрів запуску)
- Логування, створення звітів (дозвіл, в якому є звіти, можливість налаштувань, інформаційність)

При виборі інструмента, спеціалісти рекомендують взяти до увагу саму документацію застосунку, проаналізувати поставлені задачі, кваліфікацію розробників та інженерів, які відповідають за тестування, фінансові ресурси, технічних склад, плани на майбутнє, ризики [21].

1.4 Методи тестування мобільних застосунків

Тестування - це один із етапів розробки програмного забезпечення, під час якого товар або його частина проходить процес тестування, і в цей час з'ясовується, де поведінка системи відрізняється від описаної у документації.

Зараз існує багато способів експериментувати, жоден з яких не може задовольнити потреби в якісній продукції. За допомогою одного з цих методів існує висока ймовірність якісного продукту, результат буде неперевершеним і очевидним.

Однак тестування необхідно проводити на всіх етапах розробки програмного забезпечення, щоб спочатку визначити сфери інтересів. Оскільки ефективність управління ручним інспектуванням відрізняється не тільки від компанії до компанії, але й від тестувальника до тестувальника, то й виникає проблема відсутності універсального інтерфейсу тестування.

Функціональне тестування - це тестування програмного забезпечення для визначення необхідних властивостей продуктивності, тобто здатності

програмного забезпечення вирішувати проблеми, які потрібні користувачам у певних областях. Потрібно працювати над вимогами, щоб визначити, яке програмне забезпечення робити, які завдання виконувати.

Ручне тестування - це процес пошуку вад програми, коли інженер якості відповідає за всі аспекти роботи, будучи потенційним користувачем. Як правило, для вимірювання точності інженер використовує заздалегідь сплановані плани, які показують результати роботи найголовніших аспектів програми при певних операціях.

Автоматизоване тестування - це тестування, які використовують програмне забезпечення для проведення тестів та оцінки продуктивності. Автоматичне тестування надає переваги, економить час та гроші для компаній.

Існує два основних способи перевірки автоматизації: вимірювання працездатності програми на рівні коду та користувацького інтерфейсу (включаючи графічний інтерфейс користувача). Перший тип - це, зокрема, тестування по модулям. Другий - подати приклад дій користувачів, використовуючи спеціальні тестові фреймворки.

1.5 Існуючі способи автоматизації. Підходи. Паттерни.

Існує 4 основних підходи, на даний момент, до автоматизації тестування користувацького інтерфейсу:

1. Нотування і відтворення.

Вибраний існуючий спосіб характеризується використанням утиліт нотування(запису) та відтворення. Тобто, банальне клацання мишки по екрану, при тестуванні користувацького інтерфейсу, можна легко замінити автотестом. Для цього потрібно встановити потрібну утиліту, запустити режим запису та виконувати конкретні кроки під час запису. Після завершення запису утиліта запа'мятала усі необхідні кроки та тепер може відтворювати ці кроки уже без дій тестувальника, при чому, кількість повторів може бути необмеженою. Це дуже дешевий та доступний спосіб автоматизації.

2. Написання сценаріїв.

Полягає цей спосіб у написанні тестових сценаріїв існуючими на даний момент мовами програмування, такими як Java, Python, C/C++, JavaScript тощо. Розробляються спеціально для автоматизації тестування. Таким чином можна вирішити недоліки масштабування і підтримки тестів. При цьому, цей спосіб є набагато дорожчим та потребуючим постійної підтримки у порівнянні з методом нотування та відтворення.

3. Керування даними тестування.

Методологія методу полягає у створенні програмного коду для автоматизації та верифікації його на основі даних. Дані зберігаються у сховищі, онлайн або базі даних. Використовується даних підхід при певних випадках: якщо потрібно реалізувати схожі, однотипні види перевірок для багатьох варіантів вхідних даних. Спосіб є дорожчим у порівнянні із першим способом підходу до автоматизації користувацького інтерфейсу так як потребує бази даних, для роботи з якими необхідні висококласні спеціалісти.

4. Тестування, яке виконується по ключовим словам.

Методологія даного методу полягає у тестах, які являються не програмним кодом, а послідовністю певних дій з їх параметрами, яка у свою чергу описана за допомогою ключових слів. Сама ж реалізація виконується за допомогою фреймворку. Займатись таким способом підходу можуть працівники без досвіду роботи у сфері програмування, що й веде за собою недостатню ефективність для організації повноцінного тестування. Проте, даний метод отримав велику популярність через простоту та доступність.

Аналіз: для довготривалого користування, складних проєктів розробки програмного забезпечення найбільш підходящими є другий та третій методи. Перераховані способи потребують більше ресурсів для реалізації, проте з часом приносять окупність у майбутньому. Увесь написаний код можна використовувати при регулярному регресивному, частішому смоук-тестуванню, включати у процес безперервної інтеграції тощо.

При написанні тест-скриптів зазвичай застосовують технологію фреймворків. Фреймворк – це організація проєкта, яка дозволяє спростити розробку, модифікацію та підтримку програмного коду. У випадку організації проєкту по автоматизації тестування користувацького інтерфейсу програмного продукту. Основну задачу зниження кількості змін, які будуть внесені у тести вирішує саме фреймворк.

На сьогодні галузь автоматизації програмного продукту є дуже відомий шаблон проєктування, а саме – Page Object. Наслідування поточного шаблону запобігає дублюванню коду, значно спрощує підтримку тестів, їх написання, підходи. Основний концепт шаблону вміщує в себе: виділене розмежування між кодом для тестів та кодом, який є службовим і веде до елементів самого користувацького інтерфейсу, наявність єдиного репозиторію для служб та операцій.

Приклад із пояснення даного патерна відомим спеціалістом Мартіном Фаулером[22]:

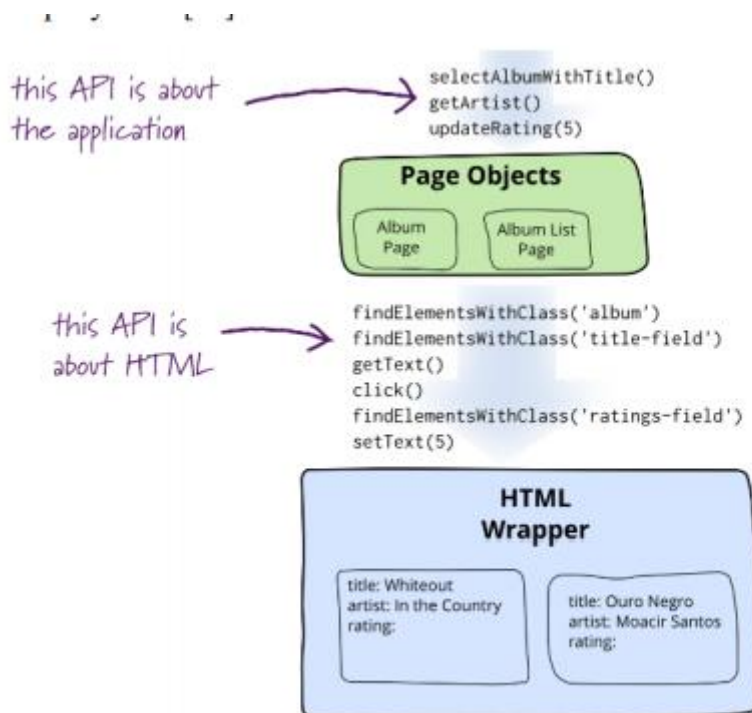


Рисунок 1.5.1 – Архітектура коду у патерні Page Object

Слід зазначити, що не тільки Page Object використовується як патерн для реалізації автоматизованого тестування програмного продукту. Відомі патерни: Screenplay, Presenter First, Порти та адаптери тощо.

Якщо провести аналогію між патернами Page Object та Screenplay, то слід бути обережним, адже з часом код, написаний за допомогою першого із перерахованих патернів може вийти із під контролю. Патерн Screenplay являє собою принцип проєктування SOLID[23] для такого виду автоматизованого тестування як приймальне. Суть патерна полягає у наступному: він бере об'єкти сторінки та розбиває їх на шматочки. Спілька тестувальників говорить про те, що це зробило їх тести більш надійними та легшими у відлагодженні.

Приклад із пояснення даного патерна відомим спеціалістом Джоном Смарт[24]:

```
@Given("^that (.*) has decided to check available tickets$")
public void decided_to_travel_by_train(String personaName) throws Throwable {
    theActorCalled(personaName).attemptsTo(
        Navigate.to(BuyTickets)
    );
}

@When("^s?he looks at a trip from (.*) to (.*) leaving (.*)$")
public void looks_at_a_trip(String origin,
    String destination,
    DepartureDay departureDay) throws Throwable {
    theActorInTheSpotlight().attemptsTo(
        FindTickets
        .forAOneWayTrip()
        .from(origin)
        .to(destination)
        .leaving(departureDay)
    );
}
```

Рисунок 1.5.2 – Архітектура коду у патерні Screenplay

Багатьом може здаватись, що цей шаблон набагато складніший у використанні, проте Джон підтвердив той факт, що використання саме патерну Screenplay заощаджує команді набагато більше часу, зменшуючи витрати часу та ресурсів на обслуговування та підтримку даного патерну.

Порти і адаптери. Архітектура спрямована на те, що є конкретний об'єкт і він зобов'язаний робити щось одне і має тільки одну причину для змін.

Відмова від використання мереж та інтерфейсу вводу/виводу, щоб нічого не сповільнювало роботу тестового артефакту.

Presenter First – модифікація для організації коду і розробки з ціллю створення повністю протестованого програмного забезпечення. До прикладу, якщо створити патерн у вигляді блок-схем, то можна побачити, що представлення є користувацьким інтерфейсом. Також можна налаштувати свою модель та контролер так, щоб імітувати всі види, які слугують для проведення, наприклад, тесту навантаження. Тобто, можна імітувати всі види «дивної» поведінки, нехай, обрив каналів зв'язку.

					ІАЛЦ.045440.004 ПЗ	Лист
						18
Зм	Лист	№ докум.	Підп.	Дата		

2. ТЕСТУВАННЯ «РОЗРОБКИ ЧЕРЕЗ ПОВЕДІНКУ» ТРАФІКУ МОБІЛЬНИХ ЗАСТОСУНКІВ

2.1. Технологія «Розробки через поведінку»

Розробка через поведінку(BDD – Behavior-Driver Development) – це метод розробки програмного продукту, є гілкою методології розробки через тестування(TDD). Гнучкий процес розробки програмного продукту, заохочує співпрацю розробників, QA і бізнес-учасників проєкту. Задумкою було розробка програмного забезпечення для розуміння між бізнес-процесами та технічною стороною, для чого було створено спеціальні інструменти, які й почали використовуватись саме для тестування [9].

Основні принципи технології, які слугують в деякій мірі головними питаннями:

- Початок процесу;
- Що перевіряється і навпаки;
- Як довго триває процес;
- Як інтерпретувати фейл тесту;
- Що викликає тест.

Описання системи архітектури застосунку у термінах експерту предметної галузі, а не програміста, і є головною ідеєю тестування «Розробки через поведінку». Це дозволяє наблизити процес отримання зворотного зв'язку до ідеалу та прибрати мовні бар'єри для аутсорсних або ж продуктових компаній, партнери яких можуть знаходитись у різних куточках планети.

З допомогою даного метода не тільки уповноважена особа в ролі програміста, який пише код, може описати принципи роботи програмного забезпечення, а й менеджер, який у свою чергу детально розуміє систему з точки зору кінцевого користувача. Тестування інтерфейсу застосунку є лишень частиною загальної задачі. Для повноцінного тестування слід врахувати усі частини: взаємодію між компонентами системи в цілому.

Результатом перевірок є сукупність запитів, які були відправлені на сервер, і відповідей, які отримав програмний продукт.

На рисунку схема показує дві області, в рамках яких реалізована одночасна перевірка функціоналу. Для перевірки інтерфейсу слугує комбінація із Cucumber, Arrium, Ruby. Для перевірки клієнт-серверної архітектури - Mock-server.

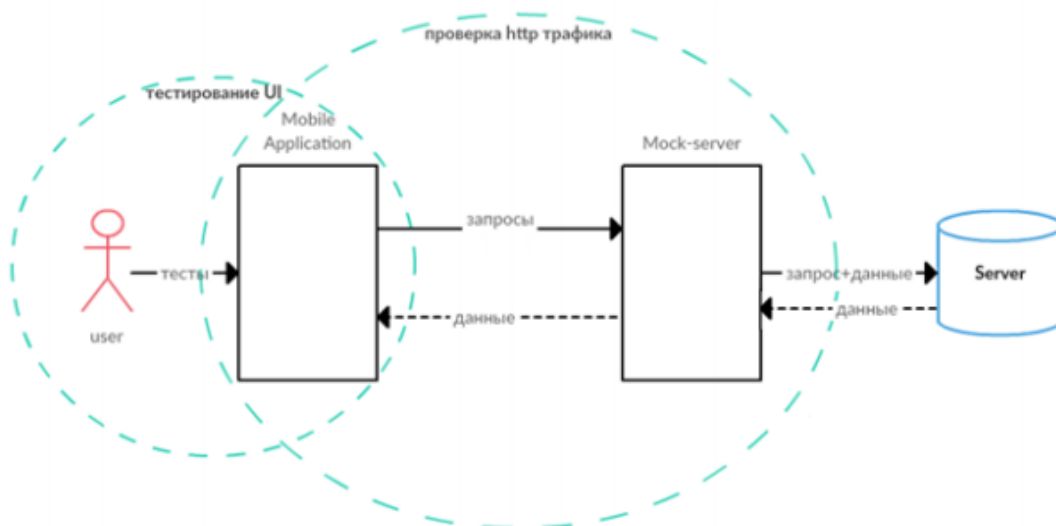


Рисунок 2.1 – Схема одночасного контролю графічного інтерфейсу користувача та мережевої взаємодії

Для аналізу запитів необхідно здійснювати перехоплення трафіку(дані від програмного продукту до серверу). В рамках такого перехоплення встановлюється формальне з'єднання, підтримується, яке завершується через спеціальний процес. Передача інформації у HTTP здійснюється тільки між сервером та додатком, тому для реалізації перехоплення трафіку здійснюється атака посередника. Будь-яка атака, по суті, є атакою перехоплення сеансу, який встановлюється між пристроями [20].

2.2 Проху

Проху – це посередник, дії здійснюються від лиця кінцевого користувача [33].

Це сервер, який взаємодіє з адресами: встановлює з'єднання та перекидує його на необхідну адресу, одночасно записуючи інформацію(логи) переданих даних.

У комп'ютерній мережі, проксі-сервер являє собою серверний додаток або пристрій, який діє як посередник для запитів від клієнтів, які шукають ресурси з серверів, які надають ці ресурси. Проксі-сервер працює від клієнта при запиті послуги, потенційно маскує справжнє походження запиту до сервера ресурсів.

Замість безпосереднього підключення до сервера, який може виконати необхідний ресурс, наприклад, на сторінці файлу веб-клієнт надсилає запит на проксі-сервер, який оцінює його і виконує необхідні мережеві угоди. Це слугує як спосіб спростити і контролювати складність запиту, або надати додаткові переваги, такі як балансування навантаження, конфіденційність і безпеку.

Веб-проксі можуть використовуватись для різних цілей: тестування сервісів, прискорення завантаження веб-матеріалів, змога оминати адміністративні обмеження локальної мережі, приховання реальної IP-адреси, отримання доступу до закритих веб-сайтів тощо.

Поділяються дані сервера на відкриті й закриті. Відкриті – проксі, до яких доступ у вільному стані для будь-якого користувача, інші ж – закритий доступ, адміністрування може бути, наприклад, по IP-адресі, підмережі тощо.

Відкритий проксі-сервер - це сервер, який дозволяє практично будь-якому вузлові мережі звертатися через себе до інших вузлів мережі. Анонімні - приховують реальні IP-адреси клієнтів і тим самим надають можливість анонімно користуватися послугами мережі Інтернет. Звичайні - не приховують нічого ні від кого і видають реальний IP і все інші типові дані. Прозорі проксі - це проксі, які не повідомляють користувачів про своє існування, таким чином користувач не знає, що доступ до ресурсу здійснюється через проксі. Для користувача процес доступу до зовнішніх ресурсів залишається

незмінним. В даному випадку, проксі, по суті, є mitm-проксі, які прослуховують з'єднання «людини посередника». Проксі є непрозорим або оголошеним, коли користувачі знають про те, що вони спілкуються через проксі, бо вони звертаються до проксі. Реверс проксі - вони ж «зворотні проксі», дозволяють забезпечити безпеку при перегляді контенту, що знаходиться в одній мережі з проксі-сервером. На відміну від прозорого проксі, який перенаправляє запити клієнтів до будь-яких серверів в Інтернеті і повертає їм результат, зворотний проксі безпосередньо взаємодіє лише з асоційованими з ним серверами і повертає відповідь тільки від них. При зверненні до зворотного проксі користувач вважає, що спілкується з реальним предметом - сервером, на якому знаходиться контент[31].

2.3 Криптографічна атака MitM

У криптографії та комп'ютерній безпеці а man-in-the-middle, monster-in-the-middle, machine-in-the-middle, monkey-in-the-middle (MITM) або person-in-the-middle (PITM) - це кібератака, при якій зловмисник таємно передає та, можливо, змінює зв'язок між двома сторонами, які вважають, що вони безпосередньо спілкуються між собою [25]. Одним із прикладів атаки MITM є активне прослуховування, коли зловмисник встановлює незалежні зв'язки з жертвами та передає повідомлення між ними, щоб вони повірили, що вони спілкуються безпосередньо між собою через приватний зв'язок, коли насправді вся розмова контролюється зловмисником. Зловмисник має можливість перехоплювати всі відповідні повідомлення, що передаються між двома жертвами, та вводити нові. Це просто за багатьох обставин, наприклад, зловмисник, що перебуває в зоні прийому незашифрованої точки доступу Wi-Fi, може вставити себе як людину посередника.

Принцип роботи: атака MitM - одна з найдавніших форм кібератаки. Інформатики шукають шляхи запобігання втручанню чи підслуховуванню комунікацій з початку 1980-х.

Атаки MitM полягають у прослуховуванні між з'єднаннями двох сторін та спостереженні або маніпулюванні трафіком. Це можливо через втручання в законні мережі або створення підроблених мереж, якими зловмисник керує. Потім компрометований трафік позбавляється будь-якого шифрування з метою викрадення, зміни або перенаправлення цього трафіку до вибраного місця зловмисника (наприклад, на сайт фішингового входу). Оскільки зловмисники можуть мовчки спостерігати або перешифровувати перехоплений трафік до передбачуваності джерела після його запису або редагування, виявити атаку може бути важко.

"Атаки MitM - це атаки, коли зловмисник фактично сидить між жертвою та законним господарем, до якого жертва намагається підключитися", - говорить Йоганнес Улліхр, декан досліджень технологічного інституту SANS.

MitM охоплює широкий спектр методів та потенційних результатів, залежних від цілей та мети. Наприклад, при зачищенні SSL зловмисники встановлюють з'єднання HTTPS між собою та сервером, проте з незахищеним з'єднанням HTTP з користувачем, що означає, що інформація надсилається у вигляді простого тексту без шифрування. Атаки Evil Twin відображають законні точки доступу Wi-Fi, але повністю контролюються зловмисниками-акторами, які тепер можуть відстежувати, збирати або маніпулювати всією інформацією, яку користувач надсилає.

Приклади формування MitM атак:

Сценарій 1: Перехоплення даних

1. Зловмисник встановлює перехват пакетів для аналізу мережевого трафіку на небезпечний зв'язок.
2. Коли користувач входить на сайт, зловмисник отримує інформацію про його з'єднання та перенаправляє на фальшивий сайт, що імітує справжній.

3. Підроблений сайт зловмисника збирає дані від користувача, які зловмисник може потім використовувати на реальному сайті для доступу до інформації цілі.

У цьому випадку зловмисник перехоплює передачу даних між клієнтом і сервером. Обдуривши користувача, що він все ще спілкується із сервером, а сервер вважає, що він все ще отримує інформацію від користувача, зловмисник може перехоплювати дані обох, а також вводити власну неправдиву інформацію в будь-які майбутні передачі.

Сценарій 2: Отримання доступу до коштів:

1. Зловмисник створює підроблений сервіс чату, що імітує службу відомого банку.

2. Використовуючи знання, отримані з даних, перехоплених за першим сценарієм, зловмисник видає себе банком і починає чат із ціллю.

3. Потім зловмисник починає чат на реальному сайті банку, видаючи себе за мішень і передає необхідну інформацію для отримання доступу до рахунку цілі.

У цьому випадку зловмисник перехоплює розмову, передаючи частини обговорення обом законним учасникам.

2.3 Інструменти для реалізації автоматизованого тестування

2.3.1 Емулятор для генерації поведінки віртуального мобільного пристрою - AVD

Віртуальний пристрій Android (AVD) - це конфігурація, яка визначає характеристики Android-телефону, планшета, Wear OS, Android TV або автомобільної ОС, які потрібно імітувати в емуляторі Android. Менеджер AVD - це інтерфейс, який ви можете запустити з Android Studio, який допомагає створювати AVD-файли та керувати ними. Android SDK включає віртуальний емулятор мобільних пристроїв, який працює на вашому комп'ютері. Емулятор дозволяє створювати зразки, розробляти та тестувати програми для Android, не використовуючи фізичний пристрій.

До плюсів використання даної утиліти відносяться:

- Крос-платформність;
- Відсутня необхідність у будь-яких сторонніх застосунках;
- Варіаційність(параметри телефону);
- Виставлення будь-якої необхідної операційної системи для смартфона.

До мінусів можна віднести:

- Повільний;
- Вимогливий до параметрів комп'ютера, на якому здійснюється запуск даної утиліти;
- Не емулює поведінку Bluetooth.

2.3.2 Genymotion

Genymotion Desktop - це емулятор Android, який включає повний набір датчиків та функцій для взаємодії з віртуальним середовищем Android[26]. За допомогою Genymotion Desktop можна тестувати свої програми Android на широкому діапазоні віртуальних пристроїв для цілей розробки, тестування та демонстрації. Genymotion Desktop швидкий, простий у встановленні та потужний завдяки зручним віджетам датчиків та функціям взаємодії. Він доступний для операційних систем Windows, macOS та Linux.

Інтелектуальне забезпечення та деініціалізація всіх робочих навантажень на вимогу є складним завданням у хмарних обчисленнях. Можливість максимально автоматизувати їх у віртуалізованих операційних системах - на хмарних платформах, таких як Amazon EC2 (AWS), допомагає досягти цього. Доступні віртуалізовані операційні системи: Linux, Windows ... Але з часів зростання мобільних додатків ще ніхто не може запропонувати професійне хмарне рішення, яке могло б автоматизувати клонування та масштабування віртуалізованої операційної системи Android.

Genymotion on-Demand - це саме така, повністю функціональна операційна система Android, доступна на Amazon EC2. За допомогою цієї

утиліти є можливість налаштувати свій екземпляр Android і - завдяки інструментам Amazon Web Services - автоматизувати його розгортання на льоту за потреби. Керування балансуванням навантаження є доволі простим і зручним.

До плюсів використання даної утиліти відносяться:

- Крос-платформність;
- Швидкість;
- Варіаційність(додаткові інструментів для контролю).

До мінусів можна віднести:

- Платне комерційне використання;
- Працює тільки при активному підключенні до інтернету.

2.3.3 Bluetacks

Bluestacks - неймовірно популярний програмний інструмент для Windows. Це один з найкращих емуляторів Android для ПК. Спочатку це був не ідеальний інструмент, але, оскільки кількість користувачів Bluestacks зростала, розробники докладали набагато більше зусиль, щоб покращити його, як сьогодні.

BlueStacks - це програмне забезпечення, яке переносить Android на комп'ютер з ОС Windows. Це дає можливість використовувати мишку та клавіатуру з вашої системи з програмами для Android і, на відміну від смартфона з екраном фіксованого розміру, можна змінити розмір вікон BlueStacks, щоб зробити програми більшими або повноекранними.

Не кожна програма для Android пропонує еквівалент Windows, macOS або веб-сервіс. BlueStacks - це простий спосіб отримати доступ до програм Android на комп'ютері з Windows.

BlueStacks може запускати кілька програм Android одночасно, і кожна програма відкривається на новій вкладці в BlueStacks. Він також пропонує багатовимірний режим, який запускає кілька екземплярів одного додатка, що

може сподобатися цінувачам продуктивності та геймерам. BlueStacks також дозволяє транслювати ігри Android в прямому ефірі на Twitch.

До плюсів використання даної утиліти відносяться:

- Крос-платформність;
- Швидкість.

До мінусів можна віднести:

- Відсутність реалізації на Linux OS.

2.3.4 MITMf

Це дуже приємний інструмент, написаний на Python, і здатний виконувати атаки людини посередині за допомогою декількох методів[27].

Головною особливістю MITMf є те, що можна обробляти пакети за допомогою Scapy - так задіюються всі знання, які набуті при вивченні цієї бібліотеки. Фреймворк містить вбудовані сервера по тим же SMB, HTTP та DNS, які піддаються керуванню та використанню великої кількості різних плагінів.

До плюсів використання даної утиліти відносяться:

- Легкість встановлення;
- Присутня можливість модифікувати активні пакети;
- Є можливість змінювати файл конфігурації при активному сеансі

утиліти;

До мінусів можна віднести:

- Робота з даною утилітою можлива лишень з сімейством Unix.

2.3.5 DroidSheep

DroidSheep - це андроїд-інструмент, розроблений Андреасом Кохом для аналізу безпеки в бездротових мережах. В основному це інструмент викрадення сеансів, який дозволяє хакерам захоплювати файли cookie сеансу через бездротову мережу. Це означає, що є можливість фіксувати профілі веб-

сеансів користувачів, які перебувають в одній мережі. Утиліта здійснює перехоплення сесій під час передачі деяких даних по з'єднанню Wi-Fi [30].

До плюсів використання даної утиліти відносяться:

- Наявність детальних пояснень та інструкцій для встановлення та користування даною утилітою;
- Легкість у встановленні.

До мінусів можна віднести:

- Відсутня підтримка роботи з HTTP;
- Для встановлення користувач операційної системи повинен бути з правами адміністратора, що є небезпечним.

2.3.6 MitmProxy

Mitmproxy - це легкий, повнофункціональний проксі-записувач. Він має інтерактивну консоль та безліч функціональних можливостей для вивчення [29]. Mitmproxy - це ваш швейцарський годинник для налагодження, тестування, дослідження конфіденційності та тест для захисту від несанкціонованого входження. Він може використовуватися для перехоплення, перевірки, модифікації та відтворення веб-трафіку, такого як HTTP, WebSockets або будь-яких інших протоколів, захищених SSL або TLS. Присутня змога впорядковувати та декодувати різні типи повідомлень, починаючи від HTML і закінчуючи Protobuf, перехоплювати конкретні повідомлення на льоту, модифікувати їх до того, як вони досягнуть місця призначення, і відтворювати їх на клієнті або сервері пізніше.

До плюсів використання даної утиліти відносяться:

- Легкість встановлювання забезпечення;
- Безкоштовне використання даного записувача;
- Можливість захоплення HTTPS.

До мінусів можна віднести:

- Працює тільки із сімейством Unix.

3. ОПИС ТЕХНОЛОГІЇ ТЕСТУВАННЯ

Розглянемо схему, яка описана нижче для розробки та тестування програмного продукту, який реалізований для мобільних телефонів на платформі Android.

Клієнтом у даному випадку є наша прикладна програма, зміни якої не відбудуться, ні в запитаннях ні в відповідях.

Після того, як клієнт відправив запит, на відповідну адресу мережі, запит перехоплюється за допомогою утиліти mitmproxy, яка встановлена на робочій станції користувача і поділена на окремі частини HTTP запиту.

Для подальшого використання зберігається адреса запиту, тип запиту, заголовки запиту та дані, що відправляються.

Далі імітуємо роботу справжнього сервера, до якого був відправлений запит. На працюючій станції налаштовується PROXY - сервер, який надсилатиме заголовлені шаблонні відповіді на запити додатку.

На справжньому сервері реалізована структура API, робота якого імітується від відправлення відповідей на запити. Щоб проаналізувати коректність роботи програми порівнюємо запити від додатку з очікуваними і запитами з серверу, які вже йтимуть з відповіддю.

3.1 Опис додатку для автоматизованого тестування

Додаток, який взятий для проведення тестування, призначений для агентів страхових компаній. Основний функціонал додатку: проведення створення та редагування звітів. Самі звіти зберігаються на хмарковому сховищі. Сервер також містить в собі інформацію про користувачів додатку, адреси, посилання тощо. Для доступу до серверу використовується захищене з'єднання у форматі HTTPS. Користувацький інтерфейс розміщений на декількох екранах, створення звіту тощо.

Потік приблизної архітектури виглядатиме приблизно наступним чином. Коли клієнт відправляє запит на адресу мережі, його перехоплює утиліта

mitmproxу, що поділена та працює на робочій станції клієнта, та розбиває на окремі HTTP-частини.

Заголовок, адреса та тип запиту, а також відправленні дані зберігаються утилітою для подальшого використання за потреби.

Наступним кроком є імітація робочого процесу справжнього сервера, що мав отримати запит.

З цією метою створюємо на робочій станції прозорий Proху та робимо заготовки відповідей. Наш сервер відправить ці заготовки у відповідь на запит додатку.

На цьому Proху-сервері реалізовано структуру API, яку найчастіше застосовують на справжніх серверах, роботу яких імітуємо для відправки реакцій на запити. Щоб перевірити коректність виконаної роботи, ефективно буде порівняти запити, що відправляв додаток, з тими, які ми очікували, а також підстановкою вирогідними відповідями реального сервера.

3.2 Cucumber

Cucumber – це інструмент тестування, який підтримує структуру поведінки (BDD). Він визначає поведінку додатків, використовуючи простий англійський текст, який визначається мовою під назвою Gherkin[28]. Це програмний засіб, з допомогою якого можна проводити тестування різного по функціональності забезпечення.

Таким чином, Cucumber дозволяє розуміти основну поведінку «під капотом» для користувача, при цьому мовою, яка є зрозумілою для усіх членів команди. Кожен тестовий сценарій у Cucumber-системі називається Feature. Всі ці так звані «фічі» лежать у папці features з специфічним розширенням *.feature. На рисунку 3.2.1 приведена детальна схема створення тестових сценаріїв.

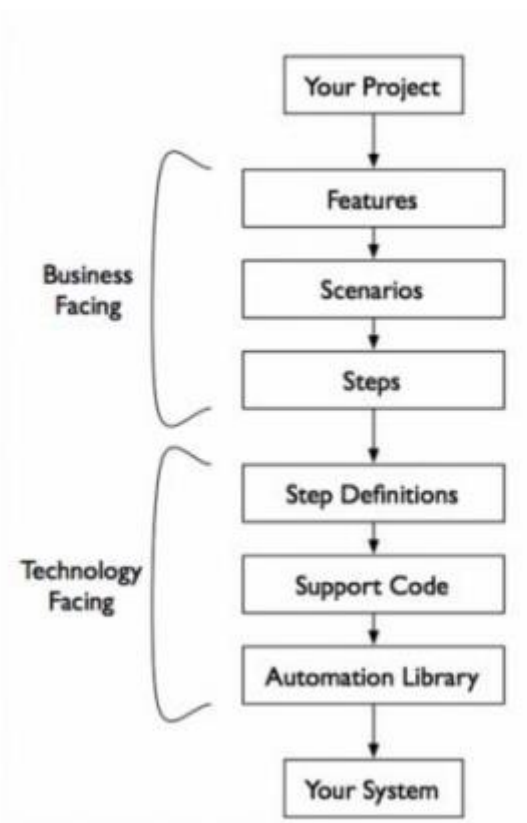


Рисунок 3.2.1 – Схема процесу написання тестових сценаріїв

Тестові випадки на Cucumber вміщують у себе зразу два файли: сам сценарій та його реалізації мовою програмування. Це дозволяє відділити очевидний опис сценаріїв від їх реалізації на незрозумілій для інших колег мові, прості англійські слова.

При безпосередньому запуску Cucumber покроково читає файл “Feature”. Насамперед, це опис очікуваного результату. Опис можна корегувати у будь-якій формі, але слід пам'ятати, що завжди потрібно починати з ключового слова Feature.

Налаштування має одне або декілька безпосередніх сценаріїв поведінки. Вони ж і складаються з певних кроків відтворення, які оголошені у файлах `step_definitions/*_steps.rb`. Таким чином Cucumber інтерпретує кожен крок та запускає TestFixtures.

```

namespace app\tests\unit\models;

use yii\codeception\DbTestCase;
use app\tests\fixtures\UserProfileFixture;

class UserProfileTest extends DbTestCase
{
    public function fixtures()
    {
        return [
            'profiles' => UserProfileFixture::className(),
        ];
    }

    // ...методы тестирования...
}

```

Рисунок 3.2.2 – приклад фікстури

В результаті отримуємо test-suite, який є написаний зрозумілою абсолютно для усіх членів команди мовою.

Наведемо деякий приклад тестового сценарію для заповнення звіту:

Feature: Complete the report

Scenario: To complete, you must first log in to the application and then create a report

Given login a@a.aa password 123qwe

And click the Add button

And I enter the value Ann in the Insured Name field

And I enter in the Property Address field the value of the Luxembourg rose

And I enter in the City field the value of Tomsk

And I put 645750 in the Zip Code field

And select Australia from the Country drop-down list

Then click Done button

Найважливішою частиною програмування та створення тестових сценаріїв є Step definitions. Саме на цьому етапі необхідно писати код, тобто

Зм	Лист	№ докум.	Підп.	Дата

ІАЛЦ.045440.004 ПЗ

Лист

32

автоматизувати процес тестування. На рисунку зображено взаємодію Features та Step Definitions.

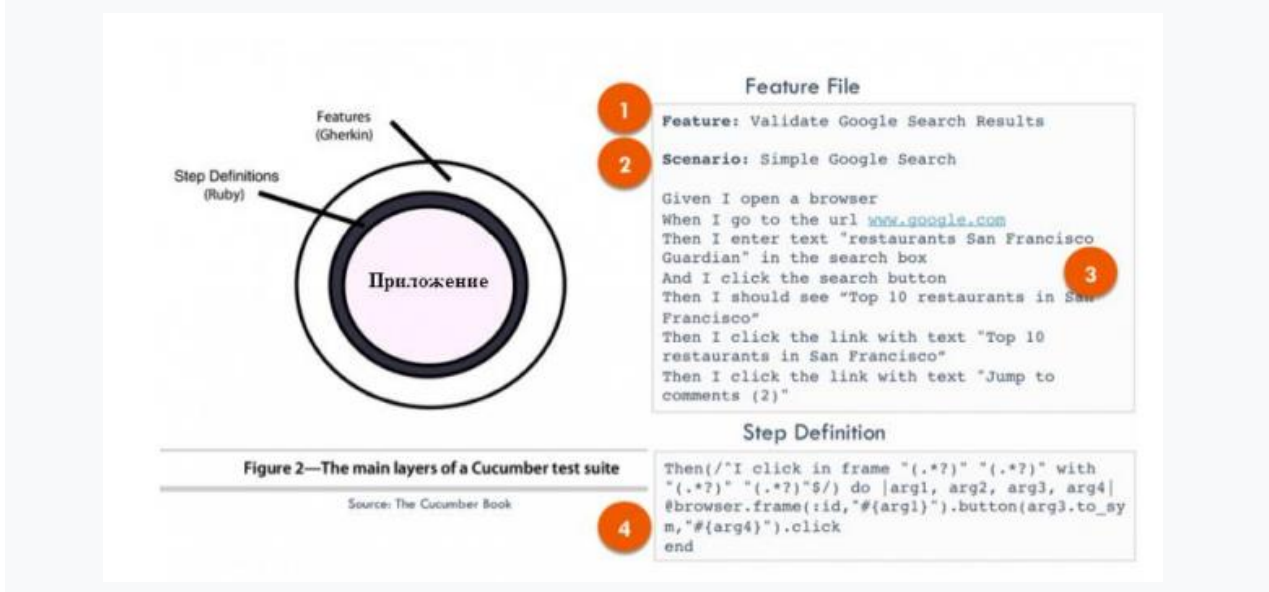


Рисунок 3.2.3– Відображення взаємодії Features та Step Definitions. 1 – Feature, 2 – Scenario, 3 – Given/When/Then, 4 – Step Definition

3.3 Appium

Щоб запустити виконання тестів з допомоги Appium, спочатку потрібно встановити усі необхідні драйвера та застосувати усі необхідні налаштування. Використовуючи Appium, не потрібно ніяк додатково модифікувати додатки, які тестуються.

Appium запускається окремим процесом, тому зразу після інсталяції усіх необхідних ресурсів є можливість одразу почати використання. Далі потрібно вказати адресу та порт, по яким й буде здійснюватись запуск Appium. Також у файлі для налаштування необхідно вказувати тестовий додаток, платформу та додаткові бібліотеки.

3.4 Mitmproxy

Mitmproxy використовується для тестування та відлагодження веб-додатків, безпосередньо з його допомогою можна отримувати інформацію про запити та відповіді, тобто про комунікацію між додатком та сервером. Також, Mitmproxy використовується в даному дипломному проєкті через прозорість роботи, тобто, проксі відкритий та є можливість здійснювати перехоплення

трафіку для тестування та аналізу, щоб захистити додатки від підозрілого з'єднання.

Mitmproxу – це утиліта, яка включає в себе ряд інструментів для роботи.

Опис:

- Mitmproxу – програма у консолі, яка здійснює перехоплення трафіку в онлайн-режимі, зберігає історію комунікації додатку із сервером, відтворює усі запити, які посилались на сервер в минулому, також відтворює відповіді сервера.
- Mitmdump – інструмент, який перехоплює трафік та паралельно записує усе в текстовий файл із усією детальною інформацією. Після завершення використання інструментом файл зберігається у відповідній директорії та у будь-який момент його можна відкрити та проаналізувати усі необхідні дані.
- Libmproxу – інструмент, який є бібліотекою, написаною мовою Python, за допомогою якої й реалізований увесь функціонал Mitmproxу. Описані усі можливості для керування, перехопленням пакетів тощо.

Назва Mitmproxу походить від MITM, що означає людина посередині. Про цю методику розказано у другому пункті другого розділу. Архітектура наступна: взломник(в даному випадку MitmProху) знаходить між клієнтським додатком та сервером, здійснюючи перехоплення трафіку як в одну так і в іншу сторону. MitmProху використовується як проксі-сервер, реєструючи увесь потік HTTP-трафіку. Також особливістю є те, що перехоплюючи трафік у обидві сторони, поточний проксі-сервер може змінювати запити та відповіді. На рисунку зображено процес встановлення з'єднання між клієнтським додатком та сервером і безпосередньо з Mitmproxу між ними.

3.5 Mock-server

Для того, щоб налаштувати роботи із API, створюємо окремий HTTP сервер із необхідним нам портом підключення та керуємо роутерами даного серверу та відповідями. Завдяки створеними правилами, описаними у

expectation, mock-server опрацьовує отримані запити зі сторони мобільного додатку. Якщо не створити необхідні правила, по яким сервер формує відповіді, то за замовчуванням сервер відповідатиме 404 – Not found.

Є ряд параметрів, які повинні бути присутніми при формуванні правильного запиту до mock-серверу:

- Path- абсолютний шлях до директорії
- Query-string – параметри, які містяться у запиті до додатку
- Header- заголовки
- Cookie-додаткові параметри, які можуть міститись у запиті до додатку
- Body-головна частина запиту.

Відповідно у відповідь на сформований запит у тілі відповіді можуть бути присутні наступні параметри:

- Status code-код, який ідентифікує помилку та локалізує її
- Body- головна частина відповіді
- Headers- заголовки
- Cookies- додаткові параметри, які можуть міститись у відповіді

3.6 Опис структури API для взаємодії з мобільним додатком

API призначений для того, щоб отримувати даних у вигляді коду, що в свою чергу слугує для автоматизації формування звітів. API дозволяє звертатись до певних елементів, уникаючи використання користувацького інтерфейсу, тобто, все відбувається «під капотом». Якщо сказати точніше, то API абстрагується від того, як саме функціонально реалізована програма.

API бібліотеки включають в себе опис безпосередньо сигнатур та семантики певних функцій. Сигнатура – це спосіб оголошення функції.

3.6.1 Взаємодія із API та її методи

Принцип дії API полягає у структурі багаторівневої ієрархії, в якій представлені елементи з однаковою структуризацією. Стандартна модель мережі OSI включає у себе 7 рівнів: фізичний, канальний, мережевий,

транспортний, сеансовий, рівень представлення, прикладний. Ця структура дозволяє використовувати функціонал нижчих API в інтерфейсі для верхніх. Це означає, що можна уникати взаємодію між автоматизованим кодом та розробленим користувацьким інтерфейсом, що спрощує задачу та виправдовує таку популярність API на сьогодні.

Якщо відправити неправильно сформований запит додатку, методи API повертають повідомлення із помилками та кодом. У реалізованій API коди помилок наступні: 403001, 404002. Перший код використовується, коли помилка виникла саме зі сторони додатку, тобто запит сформовано невірно та невідповідно до правил, другий ж інформує про те, що сервер не зміг опрацювати правильно сформований запит.

Є ряд певних функцій, які доступні через використання API:

При авторизації, новий користувач не створюється, а тільки відправляється лист на пошту. У повідомленні присутній код, який генерує API. Для реалізації цього сценарію використовується метод POST. Нижче, на рисунку наведено приклад формування запиту на авторизацію кінцевого клієнта у додаток.

```

request (application/json)

{

    "email": "john.doe@example.com", // required.

    "password": "4v5mwef3e", // required, plain text.

    "fullName": "John Doe", // required

    "facebookToken": "...", // optional

    "company": "Left Corporation Ltd." // optional

    "address": < address entity >

}

```

Рисунок 3.6.1.1– Приклад запиту API на авторизацію

Після отриманого запиту, сервер повинен сгенерувати відповідь, а саме згенерувати пакет із секретним кодом та відправити користувачеві. У випадку, коли користувач не отримує повідомлення із секретним кодом, він має змогу запросити його знову. Таким чином буде повторна генерація запиту, тобто кожен раз буде формуватись при виклику метода API. Використовується то й же метод POST:

```

request (application/json)

{

    "email": "john.doe@example.com" // required.

}

```

Рисунок 3.6.1.2– Генерація е-майлу користувача

Створення звіту відбувається на сервері та у два етапи: спочатку відбувається створення самої форми звіту, а уже потім здійснюється передача необхідних параметрів, які відповідно дозволяють заповнювати дані. Після створення звіту, приходить відповідь від сервера з кодом 200, яка означає, що звіт створено успішно. Для цього випадку також використовується метод POST, приклад зображено на рисунку 3.6.1.4.

request

```
{
    < report entity >
}
```

parameters

- + query (optional, string) ... Search pattern
- + limit (optional, integer) ... max number of returned reports
- + offset (optional, integer) ... reports offset

Рисунок 3.6.1.4– Формування звіту

Для того, щоб користувач зміг отримати звіт, який створив раніше, для взаємодії з API використовується уже інший метод, GET, який повертає необхідні нам дані, в залежності від запиту. На рисунку 3.6.1.5 зображено приклад запиту на отримання звіту.

request

```
{
    < report entity >
}
```

Рисунок 3.6.1.5– Запит на отримання звіту

Щоб змінити звіт, необхідно відправити запит, у якому будуть знаходитись параметри, які будуть замінені. Для того, щоб замінити дані у уже сформованому звіті, слід використовувати метод PATCH. На рисунку 3.6.1.6 зображено приклад PATCH запиту.

```
request
{
    < report entity >
}

parameters
+ query (optional, string) ... Search pattern
+ limit (optional, integer) ... max number of returned reports
+ offset (optional, integer) ... reports offset
```

Рисунок 3.6.1.6 – запит на редагування сформованого звіту

Також може виникнути ситуація, коли користувачі потрібно буде видалити уже створений раніше звіт. Для цього, додаток формуватиме запит і використовуючи метод DELETE звертатиметься через API до серверу. На рисунку 3.6.1.7 зображено приклад даного методу.

```
request
{
    < report entity >
}
```

Рисунок 3.6.1.7– Формування методу DELETE

Перед завантаження основної частини запиту із необхідними параметрами, сервер перевіряє, чи не є цей файл, який містить параметри,

дублікатом. Якщо файл з таким ж розміром та сумою уже присутній у сховищі, то сервер відповідає з кодом 201 – Створено.

Код помилок на поверненні необхідний для того, щоб максимально описати та локалізувати проблему. У випадку, коли запит неправильно сформований, повертається не тільки код помилки, але й відповідна інструкція. Для кожного з методів API реалізовані коди помилок. На рисунку 3.6.1.8 зображено приклад помилки для коду 404 – NOT FOUND.

Наприклад:

404 NOT FOUND

```
{  
  
  "code": 404001,  
  
  "message": "User not found"  
}
```

Рисунок 3.6.1.8– приклад помилки API

Для реалізації процесу тестування API потрібно налаштувати тестове середовище та взаємодію компонент API.

3.7 Схема роботи

Абсолютно усі дані опрацьовуються тільки після відправки запиту додатком. Водночас із запуском додатку також відбувається запуск mock-сервер.

Запуск застосунку відбувається в купі з фреймворком Appium та Genymotion. Перший ж надає API для тестових сценаріїв, які написані на Python/Ruby, другий ж – емулює тестове середовище. Mitm використовується в основному для логіки. Трапляються випадки, коли додатки використовують https з'єднання, тоді mitm, для обходу, дозволяє створювати ssl-сертифікати.

Завершення прогону тестів відбувається після того, як усі кроки будуть успішно виконані або ж виникне помилка. У будь-якому із цих випадків присутнє логування.

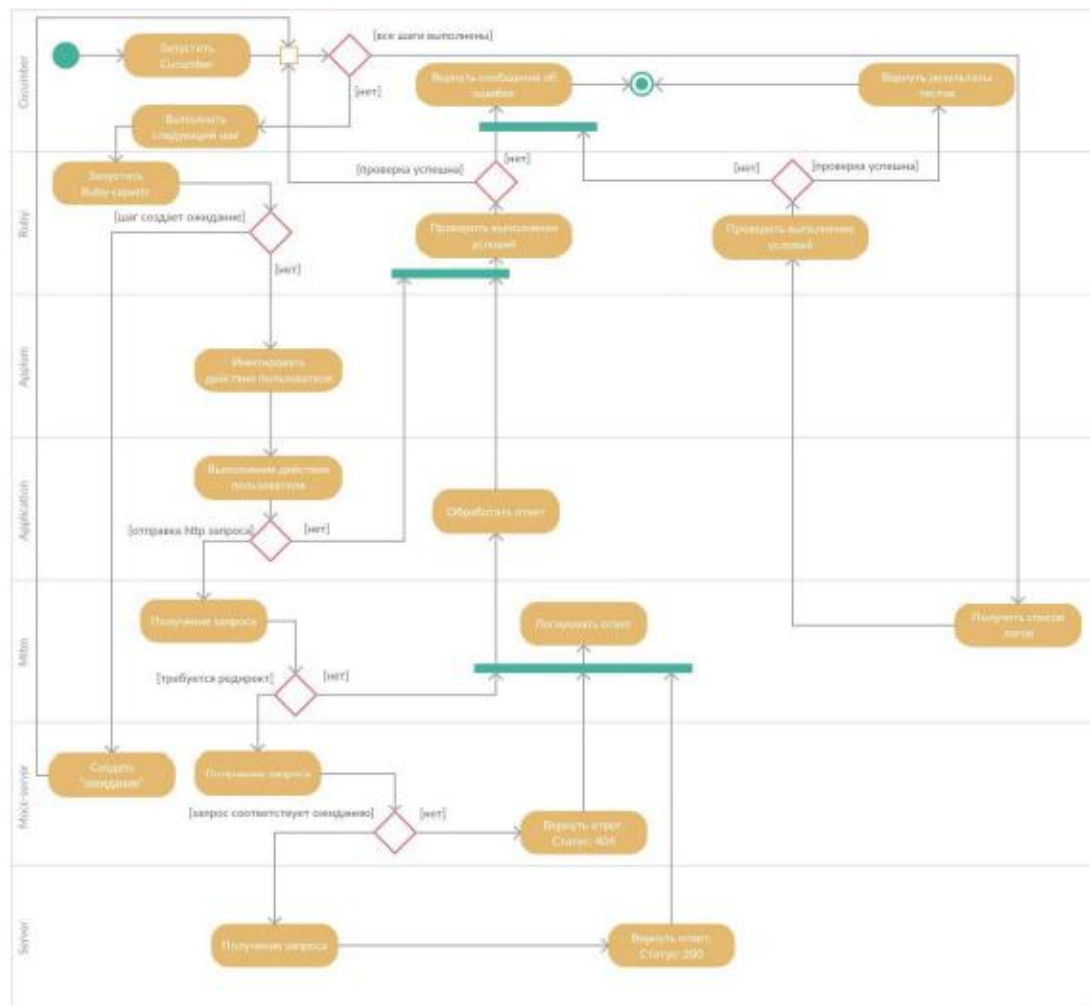


Рисунок 3.7– Принцип роботи автоматизованого тестування

4. РЕАЛІЗАЦІЯ КОМП'ЮТЕРНИХ ЗАСОБІВ ДЛЯ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ ДОДАТКІВ

4.1 Створення тестів мовою Gherkin

Написання тестових сценаріїв мовою Gherkin дає можливість виконати одразу дві задачі: отримання проєктної документації та ієрархію автоматизованого тестування [5]. Для реалізації саме процесу перевірок необхідно написати тестові сценарії, які виконуватимуться для визначення готовності програмного продукту. Ключовими термінами при використанні мови програмування Gherkin є “Features” та “Scenario”. “Features” описує приблизну поведінку тестового сценарію, тобто призначення тесту. “Scenario” описує очікувану поведінку програмного продукту в скороченому вигляді.

Приклад “Features” для входження у API зображено на рисунку 4.1.1.

```
Feature: log in  
in order login page  
A admin  
Should be go to page with document
```

Рисунок 4.1.1– Приклад “Feature” для входження у API

Особливістю мови Gherkin є те, що курсивом виділяються ключові слова, які необхідні для розуміння, проте у даному випадку вони не є обов’язковими.

Приклад “Scenario”, описуючий виконання тестів для входження на сайт:

```
Scenario:  
When user is "admin" and password is "1"  
Then response does not contain user
```

Рисунок 4.1.2 – Приклад “Scenario” для входження в API

На відміну від “Features”, ключові слова, які виділені курсивом, є обов’язковими. *When* використовується для опису дії, яка виконуватиметься

програмним кодом. *Then* описує реакцію на дії, та описує правильну поведінку програмного продукту. *Scenario* – ключове слово, яке вказує на початок сценаріїв в скороченому вигляді.

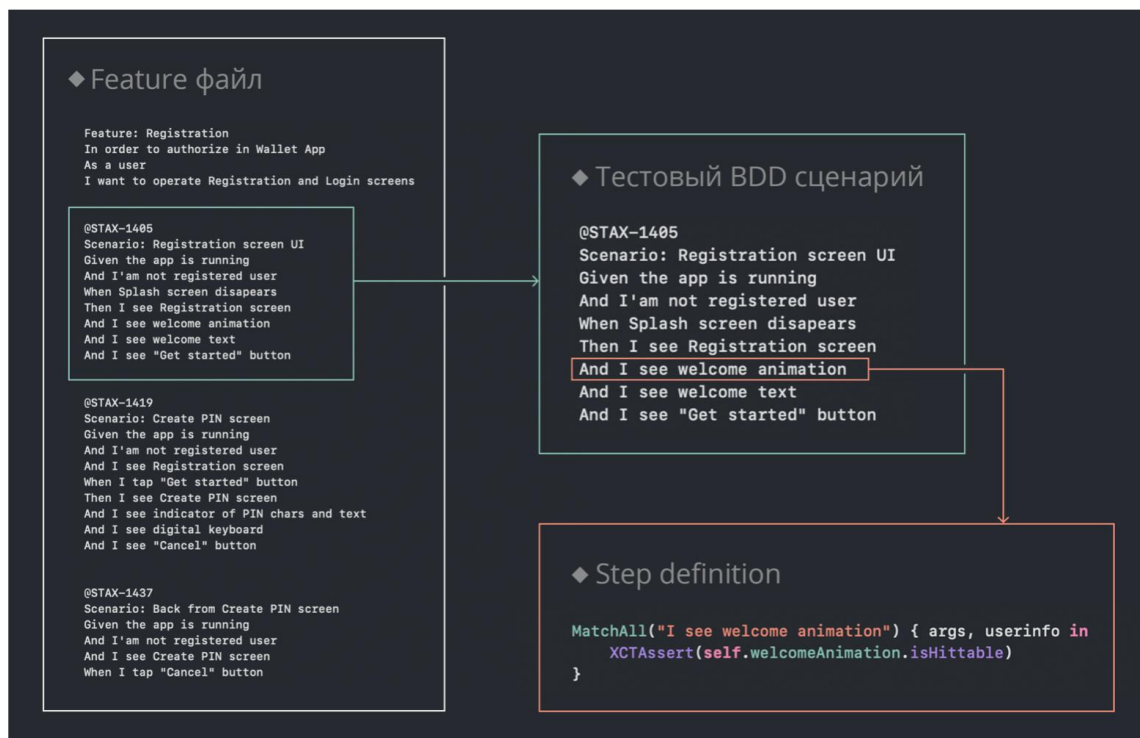


Рисунок 4.1.3– Потік виконання тестових сценаріїв

На рисунку 4.1.3 зображено загальну ієрархію виконання автоматизованого тестування за допомогою Gherkin та Cucumber.

Опис поточної діаграми:

- Фреймворк Cucumber збирає даний проєкт на пристрої;
- фреймворк проходить построчно по програмному коду та по черзі запускає сценарії із першого фіча-файлу;
- фреймворк переходить на корінний файл, аналізує код, та запускає застосунок, який тестуватиметься. Фрагмент коду зображено на рисунку 4.1.4.

```
Given("the app is running") { (args, userInfo) -> Void in
    application.launch()
}
```

Рисунок 4.1.4– фрагмент коду запуску застосунку

- після проходження всіх Given кроків і виконання програмного коду дійде до місця розташування першого When або Then, попередні умови вважатимуться виконаними. За подальшими інструкціями фреймворк знову повертається до корінного файлу, де знаходяться Step Definitions;
- після завершення усіх тестових сценаріїв застосунок закривається. Далі знову береться наступний сценарій з корінного фіче-файла, відбувається виконання сценарію і так далі до тих пір, поки усі сценарії не будуть вичерпаними.

4.2 Розробка Steps to reproduce

Для того, щоб почати розробку Steps to reproduce, у створеній раніше кореневій папці створюються файли з відповідними назвами, у яких буде описаний виконуваний програмний код. Після чого, у файлі пишеться код для кожного кроку. На рисунку 4.2.1 зображено приклад написання тестового коду.

```
switch_steps $ cucumber -p android
Using the android profile...
Feature: Hi!

Scenario: Hi from platform # features/hello.feature:3
  When I am greeted # features/step_definitions_android/greet_android.rb:1
  Then I see the platform name # features/step_definitions_android/greet_android.rb:5

1 scenario (1 passed)
2 steps (2 passed)
0m0.027s
switch_steps $ cucumber -p ios
Using the ios profile...
Feature: Hi!

Scenario: Hi from platform # features/hello.feature:3
  When I am greeted # features/step_definitions_ios/greet_ios.rb:1
  Then I see the platform name # features/step_definitions_ios/greet_ios.rb:5

1 scenario (1 passed)
2 steps (2 passed)
0m0.026s
```

Рисунок 4.2.1– Приклад коду для реалізації Steps to reproduce

Також відомо, що Cucumber запускає тільки ті тестові сценарії, які відповідають регулярному виразу, тобто обов’язково повинні бути присутні ключові слова: When, Then, Given. Якщо у одному тесті буде декілька функцій, які починаються однаково, то Cucumber запустить тільки той, який підходить по регулярному виразу.

Cucumber запускає ті кроки, які є задовільними та йдуть після ключового слова: When, Then, Given. Якщо у тесті буде декілька функцій, які починаються, нехай, із однакового ключового слова, проте різним вмістом, то запуститься тільки та функція, яка підходить під регулярний вираз.

Після того як виконання тестового сценарію закінчено, здійснюється виведення висновку у консолі з відповідним результатом.

```

C:\Windows\system32\cmd.exe
C:\Users\Yui\Documents\Aptana Studio 3 Workspace\test>cucumber
*** WARNING: You must use ANSICON 1.31 or higher (https://github.com/adoxa/ansicon/) to get coloured
output on Windows
@api
Feature: log in
  In order login page
    As admin
      Should be go to page with document

  Scenario:
    When user is "admin" and password is "1" # features/feature_add_folder.feature:7
    Then response does not contain user      # features/step_definitions/login_steps.rb:5
    # features/step_definitions/login_steps.rb:12

1 scenario (1 passed)
2 steps (2 passed)
0m0.413s
C:\Users\Yui\Documents\Aptana Studio 3 Workspace\test>
  
```

Рисунок 4.2.2– Висновки після виконання тестів у комп’ютерній консолі

4.3 Перевірка та запуск комп’ютерних засобів

Для того, щоб виконання тестів з використанням фреймворку Cucumber було можливим, необхідно запустити саму середовище для тестових випадків.

Середовище включатиме в себе ряд компонент:

- Appium
- Genymotion
- Mock-server

Appium - це вільно розповсюджена платформа тестування інтерфейсу мобільного додатка з відкритим кодом. Appium дозволяє тестувати власні, гібридні та веб-додатки і підтримує тест автоматизації на фізичних пристроях, а також на емуляторі або симуляторі. Він пропонує тестування додатків на різних платформах, тобто єдиний API працює як для тестових сценаріїв платформи Android, так і для iOS[32].

Genymotion Desktop - це емулятор Android, який включає повний набір датчиків та функцій для взаємодії з віртуальним середовищем Android. За допомогою Genymotion Desktop ви можете тестувати свої програми для

Android на широкому діапазоні віртуальних пристроїв для цілей розробки, тестування та демонстрації. Genymotion Desktop швидкий, простий у встановленні та потужний завдяки зручним віджетам датчиків та функціям взаємодії. Він доступний для операційних систем Windows, macOS та Linux.

Щоб запустити середовище та програмний продукт, використовується емулятор Genymotion, який створює пристрій з емульованою поведінкою. Далі, на пристрої запускається тестовий додаток. Після цього можна запускати тести.

4.4 Запуск Мок-серверу. Симуляція роботи API-серверу

Після того, як середовище та додаток готові для тестів, виконання тестів запущено, потрібно запустити Мок-сервер, який імітуватиме роботу API сервера. Заглушка виконуватиме такі функції: відповідатиме на запити програми, а також реалізовувати відстеження даних, що відправляються емулятором пристрою під час виконання тестів. Отриманням даних займається mock-server, що перехоплює всі запити. Для того, щоб імітований сервер зміг правильно працювати із пакетами даних та отримував їх, необхідно реалізувати певні правила, по яким імітований сервер і працюватиме. Правила оформляються у вигляді виконуваного файлу формату «.rb». У файлі вказуються заголовки запиту. Заголовки: метод, шлях, тіло запиту тощо. Правила слугують у деякій мірі ідентифікаторами для сервера, щоб розуміти, що є що.

```

include MockServer
include MockServer::Model::DSL

def create_mock_client
  MockServer::MockServerClient.new(ENV.fetch("MOCK_SERVER_HOST", 'localhost'), ENV.fetch("MOCK_SERVER_PORT", 1080))
end

let(:query_string_parameters) { [] }
let(:movies_resp_body) { '[]' }

before :each do
  @movies_server = create_mock_client
  @movies_server.reset

  @exp = expectation do |exp|
    exp.request do |request|
      request.method = 'GET'
      request.path = '/movies'
      request.headers << header('Accept', 'application/json')
      request.query_string_parameters = query_string_parameters
    end

    exp.response do |response|
      response.status_code = 200
      response.headers << header('Content-Type', 'application/json; charset=utf-8')
      response.body = body(movies_resp_body)
    end
  end

  @movies_server.register(@exp)
end

```

Рисунок – Приклад setup mock. Тіло відповідей

Для перевірки правильності посилання запитів на імітований сервер, необхідно покрити тестовими сценаріями цю ділянку, тобто end-point, з якого здійснюється підключення через API до серверу.

4.4.1 Перевірка послідовності запитів

Щоб зрозуміти, чи правильно здійснюються послідовність запитів від додатку до mock серверу, слід розуміти, що можна використати наступний метод, зображений на рисунку:

Якщо задати саме такий метод, можна забезпечити хоча б одноразову відправку запиту. Підтвердження того, що запит надійшов та був оброблений здійснюється mock сервером.

Mock-сервер очікує на вхідні запити. Якщо запит, який надійшов, підходить по критеріям, які описані у правилах на рисунку, то сервер надсилає відповідь відповідно до тих ж правил та направляє запит далі на реальний сервер.

Зм	Лист	№ докум.	Підп.	Дата

ІАЛЦ.045440.004 ПЗ

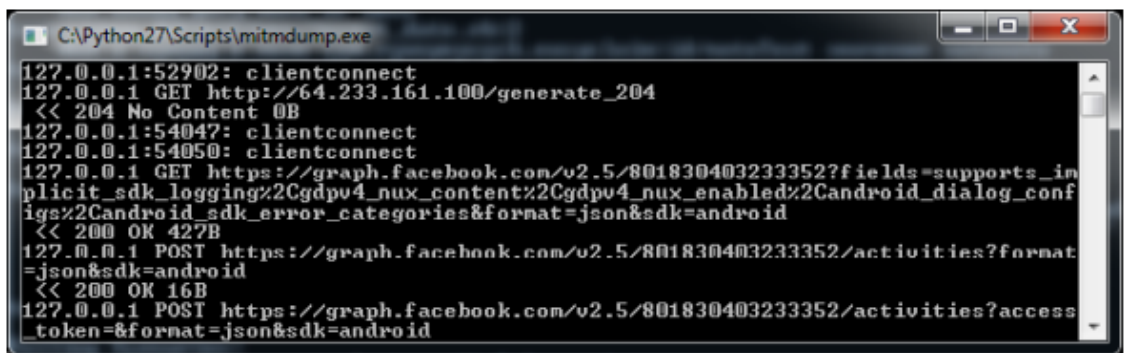
Лист

47

Запуск tosk-серверу відбувається через консольне вікно. Після запуску серверу, запускаються правила. Далі, сервер очікуватиме запити з параметрами, які потім буде порівнювати з правилами, для аналізу, чи правильно сформовані дані запиту. У випадку, коли дані неспівпадають, сервер відправлятиме помилку з кодом “Not found”.

4.5 Виконання роботи комп’ютерних засобів. Перехоплення трафіку

При незахищеному з’єднанні між сервером та клієнтським додатком mitmproху пропускає з’єднання, тобто встановлює його. Залежності між тим, чи клієнт є браузером або ж додатком – немає. Mitmproху відтворює та показує інформацію про з’єднання у консольному вікні, відповідно до рисунку 4.4.1 , або у файлі з розширенням “.txt”, далі повертає відповідь клієнтові на відповідний запит.



```
C:\Python27\Scripts\mitmdump.exe
127.0.0.1:52902: clientconnect
127.0.0.1 GET http://64.233.161.100/generate_204
<< 204 No Content 0B
127.0.0.1:54047: clientconnect
127.0.0.1:54050: clientconnect
127.0.0.1 GET https://graph.facebook.com/v2.5/801830403233352?fields=supports_in
plicit_sdk_logging%2Cgdpv4_nux_content%2Cgdpv4_nux_enabled%2Candroid_dialog_conf
igs%2Candroid_sdk_error_categories&format=json&sdk=android
<< 200 OK 427B
127.0.0.1 POST https://graph.facebook.com/v2.5/801830403233352/activities?format
=json&sdk=android
<< 200 OK 16B
127.0.0.1 POST https://graph.facebook.com/v2.5/801830403233352/activities?access
_token=&format=json&sdk=android
```

Рисунок 4.4.1 – Відображення інформації про підключення

Також можна зіткнутись з проблемою, коли з’єднання між сервером та клієнтом є захищеним. Для перехоплення саме трафіку під захистом, потрібно створити спеціальний сертифікат для mitm та додати його у директорію ~ / .mitmproху. Далі, створюючи зв’язок із браузером, другий буде визначати створені сертифікати як підозрілі, генеруючи при наступних спробах відповідні попередження, використовуючи mitmproху. Для того, щоб мати змогу відхилятися від попереджень, є можливість створити сертифікат від mitmproху у список усіх сертифікатів, які будуть використовуватися браузером при перевірці.

Процедура перехоплення складається з наступних кроків:

1. Між додатком та mitmproxу встановлюється з'єднання зі сторони додатку;
2. Сервер відповідає додатку кодом 200, що означає, що з'єднання було успішно встановлено;
3. Взаємодія між додатком та mitmproxу ідентична, як і з віддаленим сервером, тому, між ними встановлюється SSL-з'єднання. Використання Server Name Indication забезпечує правильне вказання імені хоста;
4. Mitmproxу підключається до сервера і встановлює SSL-з'єднання, використовуючи вказане додатком ім'я хоста;
5. У відповіді сервер передає SSL-сертифікат, що містить значення параметрів, на основі яких потім буде створений підмінний сертифікат;
6. Mitmproxу генерує підмінний сертифікат і продовжує SSL-діалог з клієнтом, який був загальмований на етапі 3;
7. Клієнт відправляє запит через встановлене SSL-з'єднання;
8. Mitmproxу передає запит серверу через SSL-з'єднання, встановлене на етапі 4.

4.6 Застосування розроблених комп'ютерних засобів

Сьогодні існує безліч методів для реалізації автоматизованого тестування, проте, даний метод, описаний у дипломному проєкті, забезпечує максимально короткі проміжки часу для перевірки правильності роботи програмного продукту та коректність роботи API серверу. Організація автоматизованого тестування цим методом сприяє покращенню взаємодії між усіма членами проєкту, які працюють над програмним продуктом. Бізнес-аналітики, які ставлять цілі, програмісти, які виконують код відповідно до цілей, тестувальники, які виконують перевірки правильності роботи по вимогам, тестувальники по автоматизації, які сприяють зменшенню рутинної роботи тестувальникам за допомогою BDD підходу підвищують взаємодію та розуміння між усіма перерахованими вище в рамках проєкту.

Підхід успішно використаний для забезпечення високо-якісної перевірки роботи додатків. У автоматизованому тестуванні задіяно два підходи: перший забезпечує повноцінний об'єм тестових сценаріїв на мові програмування Gherkin, які є доступними для розуміння абсолютно усім учасникам розробки програмного продукту, єдиним нюансом є володіння англійської мови та другий, який реалізує розробку тестів мовою програмування Ruby, здійснює безпосередньо перехоплення трафіку від додатку, який тестується до імітованого серверу, перевірку самого серверу. Перевірка запитів до API від тестового додатку допомагає забезпечити отримання правильної інформації зі сторони сервера, уникає зависання мобільного додатку.

Всесвітня павутина «Інтернет» заснована на архітектурі, де посиляються запити до API серверів REST. Якщо реалізувати автоматизоване тестування саме цим методом, то можна гарантувати правильну роботу багатьох додатків, запити до сервера яких реалізовані саме REST архітектурою. Підхід гарантує протестувати запити та вирішити ряд проблем:

- Формат тіла запиту:

Тестування формату тіла запиту забезпечує використання способів представлення дат, перевірку того, що всі поля є проідентифікованими, стандартизацію форматування даних.

- Формат тіла відповіді:

Тестування формату тіла відповіді допомагає знайти наступні помилки: типизація, використання саме масивів, а не складних конструкцій для передачі певних даних.

- Обробка помилок:

Тестування забезпечить те, що сервер відповідатиме коректним кодом, де буде деталізовано та локалізовано помилку.

- Безпека:

Тестування забезпечить проаналізувати, що всі комунікації між додатком та сервером здійснюються по захищеному протоколу HTTPS.

- **Продуктивність:**

Проводячи тестування продуктивності, можна проаналізувати, скільки потенційних користувачів зможуть користуватись додатком без виявлення неправильної роботи.

Використовуючи засоби, описані у даному дипломному проєкті, можна вирішити проблеми комунікацій між замовником та командою розробників та забезпечити якісне тестування програмного продукту в умовах, коли змінюються вимоги замовника. Фокус методології BDD зосереджений саме на поведінці програмної розробки, а написання тестових сценаріїв на зручній для розуміння мови програмування дозволяє узгоджувати правильну поведінку додатку з замовником, що веде за собою економію часу на розробку.

У рамках даного дипломного проєкту було розроблено тестові сценарії, написані мовою Gherkin. Розроблено модуль інтеграції та виконана реалізація тестових сценаріїв. Головною особливістю є реалізації перехоплення трафіку від додатку, який тестується, до сторонніх серверів, або ж перенаправлення до mock-server, який є імітацією поведінки реального серверу, дозволяє формувати необхідні відповіді на запити від застосунку.

Реалізовано автоматизований запуск усіх комп'ютерних засобів. За допомогою файлу з розширенням .bat можна запустити усе самостійно. На рисунку 4.6.1 показано запуск усіх тестів та фрагмент мобільного додатку у емуляторі. Використовується версія Android 10.0, девайс Pixel 3A від компанії Google. Емулювати мобільний девайс вдалось з допомогою Android Studio SDK.

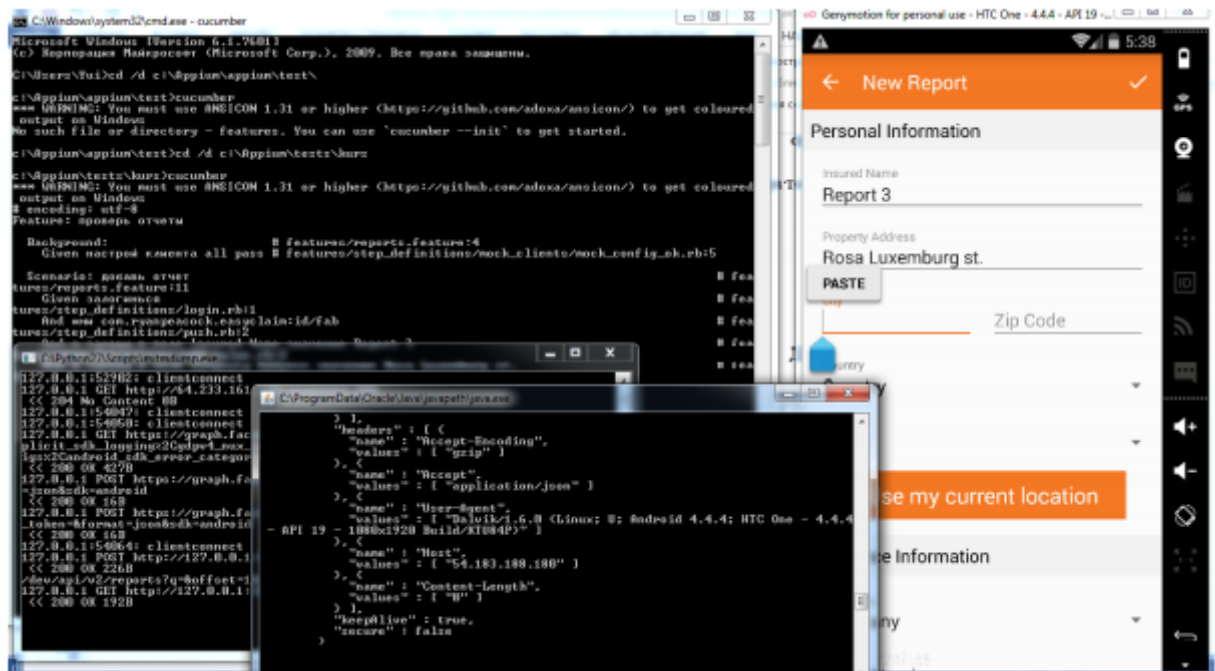


Рисунок 4.6.1 – Консольне відображення запущених тестів

Для вхідних даних, зображених на рисунку 4.6.1 запуск автоматизованих тестів показав наступні результати, зображені на рисунку 4.6.2

```
steps
-----
/I have a working Gherkin environment/ (SanitySteps.swift:17) - pass
/I use the example Insured name (? :Report 3)/ (SanitySteps.swift:38) - pass
/I use Property Address(? :Rosa Luxemburg st.) - pass
/I use City (? :Rosa) - pass
/The Zip Code is (? :355) - failed
XCTestCase+Gherkin.swift:165: error: -[XCTest_Gherkin_Tests.ExampleFeatures
testBasicSteps] : failed - Step definition not found for 'The Zip Code is'
```

Рисунок 4.6.2 Результат автоматизованого тестування

За допомогою певного набору тестових сценаріїв вдалось зрозуміти, що значення 355 для Zip-Code поля у додатку є невалідним і сервер відбив запит від додатку, бо не пройшов валідацію. Якщо заповнити усі поля правильно та спробувати створити звіт у мобільному додатку, то після запуску автоматизованих тестів можна буде зрозуміти, чи все спрацювало як слід. Приклад успішного виконання тестових сценаріїв зображено на рисунку 4.6.3

steps

```
-----  
/I have a working Gherkin environment/ (SanitySteps.swift:17) - pass  
/I use the example Insured name (:Report 3)/ (SanitySteps.swift:38) - pass  
/I use Property Address(:Rosa Luxemburg st.) - pass  
/I use City (:Rosa) - pass  
/The Zip Code is (:355) - pass  
/The Country is (: Luxemburg) - pass  
/The age should be ([18-100]*)/ (SanitySteps.swift:44) - pass  
/This is another step/ (SanitySteps.swift:33) - pass  
/This step should call another step/ (SanitySteps.swift:28) - pass  
/This test should not ([a-zA-Z0-9]*)/ (SanitySteps.swift:23) - pass  
XCTestCase+Gherkin.swift:165: successfully. finished work: -[XCTest_Gherkin_Tests  
ExampleFeatures testBasicSteps]
```

Рисунок 4.6.3 – Успішне виконання автоматизованих тестів

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Automation testing. URL:<https://smartbear.com/learn/automated-testing/best-practices-for-automation/>.
2. How do use API. URL:<https://www.coindesk.com/learn/ethereum-101/ethereum-smart-contracts-work>.
3. Lessons Learned in Software Testing/T. Pettyhort – Nevada, 2001. - 300 с.
4. Software Testing Techniques – Москва, Бейзер Б. 1983. – 284 с.
5. Software Testing Automation Tips: 50 Things Automation Engineers Should Know, Nikson, 2002. – 286с.
6. Экстремальное программирование. Разработка через тестирование
7. The PRISMA approach – Practical Risk Based Testing – Erik van Veenendaal, 2002. – 147с.
8. The Way of the Web Tester – A beginners guide to Automating tests – Jonathan Rasmussen, 1998. - 5с.
9. Java for Testers – Alan Richardson, 2019. - 89с.
10. Agile Testing, a practical guide for testers and agile teams. – Lisa Crispin/Janet Gregory, 2010. – 58с.
11. 30 Things every software tester should learn - Heather Reid, 2020.
12. Performance Testing with JMeter3 - Bayo Erinle, 2011. – 56с.
13. TED Talks, the official TED guide to public speaking – Chris Anderson. 1983. – 586с.
14. Loyalty and Disloyalty, a leader’s manual – Dag Heward-Mills, 1976. – 504с.
15. Testing according to TMap – Martin Pol, 2018.
16. ГОСТ 28806-90. URL: <http://vsegost.com/Catalog/10/10605.shtml>

					ІАЛЦ.045440.004 ІЗ	Лист 54
Зм	Лист	№ докум.	Підп.	Дата		

17. Дейкстра Дисципліна програмування; пер. З англ. І. Х. Зусман; ред. Е. З. Любимський. – М.: Світ, 1978.-275с.
18. L. Copland. A Practitioner's Guide to Software Test Design. - London: STQE Publishing, 2004. - 270 с
19. С. Куліков. Тестування програмного забезпечення. Базовий курс. - Мінськ: Чотири четверті, 2015. - 293 с.
20. Піраміда автоматизації та інші геометричні фігури // AT.Info. URL: <http://automated-testing.info/t/piramida-avtomatizacz...>, Дата звернення: 13.05.2021.
21. Д. Хіміон. Аналіз інструментів автоматизації мобільного тестування. // SQA Days-21. URL: <http://sqadays.com/ru/talk/43383>, дата звернення: 12.05.2021.
22. Автоматизація тестування: вибір інструменту. // OpenQuality.ru. Якість програмного забезпечення. URL: <http://blog.openquality.ru/tool-choice/>, дата звернення: 12.05.2021.
23. Page Object // martinowler.com. URL: <https://martinowler.com/bliki/PageObject.html>, дата звернення: 12.05.2021.
24. SOLID Tutorial. URL: <https://sites.google.com/site/unclebobconsultingllc/getting-a-solid-start>
25. Screenplay Tutorial. URL: <https://writersstore.com/blogs/news/how-to-write-a-screenplay-a-guide-to-scriptwriting>
26. BDD technology. URL: <https://encyclopedia2.thefreedictionary.com/Monkey-in-the-middle+attack>
27. Genymotion. URL: <https://4pda.to/forum/index.php?showtopic=487687>
28. MITMf. URL: <https://kali.tools/?p=179>
29. Cucumber tutorial. URL: <https://cucumber.io/docs/guides/10-minute-tutorial/>
30. Mitmproxy. URL: <https://mitmproxy.org/>
31. Droidsheep tutorial. URL: <https://spy-soft.net/instrukciya-po-ustanovke-i-nastrojke-droidsheep-android/>
32. All of proxy. URL: <https://proxys.io/ru/blog/proksi-info/razlichie-mezhdu-vpn-i-proksi>
33. Розробка через поведінку. URL: <https://conferences.vntu.edu.ua/index.php/mn/mn2020/paper/viewFile/10541/8834>