МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІМ. Ю.М. ПОТЕБНІ

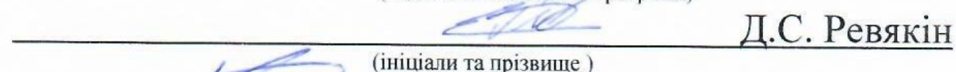
КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
АВТОМАТИЗОВАНИХ СИСТЕМ

Кваліфікаційна робота

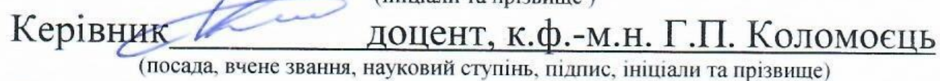
другий (магістерський)
(рівень вищої освіти)

на тему Appium як сучасний інструмент автоматизації мобільних застосунків

Виконав: студент 2 курсу, групи 8.1210-2іпз
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Інженерія програмного забезпечення
(код і назва освітньої програми)



Д.С. Ревякін
(ініціали та прізвище)



Керівник доцент, к.ф.-м.н. Г.П. Коломоєць
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ «Дісітел»

П.О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІМ. Ю.М. ПОТЕБНІ

Кафедра програмного забезпечення автоматизованих систем
Рівень вищої освіти другий (магістерський)
Спеціальність 121 Інженерія програмного забезпечення
(код та назва)
Освітня програма Інженерія програмного забезпечення
(код та назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри ________ В.Г. Вербицький
" 01 " вересня 2021 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Ревякіну Дмитру Сергійовичу
(прізвище, ім'я, по батькові)

1.Тема роботи Appium як сучасний інструмент автоматизації мобільних застосунків

керівник роботи Коломоєць Генадій Павлович, к.ф.-м.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від "30" червня 2021 року № 974-с

2. Строк подання студентом кваліфікаційної роботи 30.11.2020
3. Вихідні дані магістерської роботи
   - комплект нормативних документів ;
   - технічне завдання до роботи.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
   - огляд та збір літератури стосовно теми дипломної роботи;
   - дослідження предметної області;
   - огляд та аналіз існуючих аналогів;
   - створення програмного продукту та його опис;
   - перелік вимог в роботі з програмою та системою;
   - реалізація та тестування створеного продукту.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) 15 слайдів презентації

6. Консультанти розділів магістерської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата
		Завдання прийняв

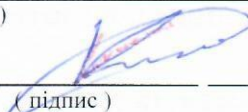
7. Дата видачі завдання 01.09.2021

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Відмітка про виконання
1	Автоматизоване тестування мобільних застосунків. Порівняльний аналіз інструментів автоматизованого тестування мобільних застосунків.	01.09-15.09	Виконано
2	Загальна характеристика Appium, архітектура Appium. Опис інсталяції Appium та підготовки середовища для тестування мобільних додатків для Android та IOS, включаючи програмні емулятори.	15.09-20.10	Виконано частково
3	Аналіз функціоналу Інтернет магазину "Елітан", складання тест-плану та розробка тестових сценаріїв для Android та IOS.	20.10-01.11	Виконано частково
4	Аналіз відмінностей інсталяції та налагодження середовищ для тестування під Android та IOS. Відмінності Desired capabilities. Аналіз ефективності тестування - використання ресурсів.	01.11-22.11	Виконано частково
5	Оформлення дипломної роботи	23.11-30.11	Виконано

Студент ________________ Д.С. Ревякін

(підпис) (прізвище та ініціали)

Керівник роботи  Г.П. Коломоєць

(підпис) (прізвище та ініціали)

Нормоконтроль пройдено

Нормоконтролер І.А. Скрипник

(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок: 90

Рисунків: 21

Таблиць: 4

Джерел: 12

Ревякін Дмитро Сергійович. Appium як сучасний інструмент автоматизації мобільних застосунків.

Кваліфікаційна робота для здобуття ступеня вищої освіти магістра за спеціальністю 121 — Інженерія програмного забезпечення, науковий керівник Г. П. Коломоєць. Інженерний навчально-науковий інститут Запорізького національного університету. Факультет енергетики, електроніки та інформаційних технологій, 2021.

Мета і завдання дослідження полягає у вивченні засобів тестування мобільних застосунків фреймворку Appium, особливостей організації тестування наітивних, веб та гібридних мобільних застосунків, розробки оптимізованої методики створення тестів із використанням інструментів Selenium.

У процесі дослідження виконано порівняння можливостей сучасних інструментів тестування мобільних застосунків. Детально вивчені засоби фреймворку Appium для тестування наітивних, веб та гібридних мобільних застосунків. Досліджені особливості запуску тестів на фізичних пристроях та емуляторі Genymotion. Розроблений тест-план для тестування мобільної версії Інтернет-магазину elitan.com.ua. Розроблені та реалізовані на мові програмування Java із використанням тестового фреймворку Junit тест-кейси для операційної системи Android, що реалізують тести мобільної версії Інтернет-магазину.

Ключові слова: *автоматизоване тестування, junit, appium, desired capabilities, selenium webdriver, selenium ide*

SUMMARY

Pages: 90

Pictures: 21

Tables: 4

Sources: 12

Revyakin Dmitry Sergeevich. Appium as a modern tool for automating mobile applications.

Qualification work for a master's degree in specialty 121 - Software Engineering, research supervisor GP Kolomoyets. Engineering Educational and Research Institute of Zaporizhia National University. Faculty of Energy, Electronics and Information Technology, 2021.

Purpose: the purpose of the study is to study the means of testing mobile applications of the Appium framework, the features of the organization of testing of native, web and hybrid mobile applications, the development of optimized methods for creating tests using Selenium tools.

Results: a comparison of the capabilities of modern tools for testing mobile applications. Appium framework tools for testing native, web and hybrid mobile applications have been studied in detail. The peculiarities of running tests on physical devices and the Genymotion emulator have been studied. A test plan has been developed for testing the mobile version of the elitan.com.ua online store. Developed and implemented in the Java programming language using the test framework Junit test cases for the Android operating system, which implement tests of the mobile version of the online store.

Keywords: automated testing, junit, appium, desired capabilities, selenium webdriver, selenium ide

ЗМІСТ

ВСТУП 8
РОЗДІЛ 1 ІНСТРУМЕНТИ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ МОБІЛЬНИХ ЗАСТОСУНКІВ 14
1.1 Історія розвитку тестування програмного забезпечення 14
1.2 Поняття автоматизованого тестування 16
1.3 Бар’єри автоматизованого тестування 18
1.3.1 Список Брета Петтіхорда 19
1.3.2 Позиція програмістів «Навіщо автоматизувати» 19
1.3.3 Початкові інвестиції 20
1.3.4 Плинність коду 21
1.3.5 Успадкований код 22
1.3.6 Страх 22
1.3.7 Старі звички 23
1.3.8 Чи можливо подолати бар’єри? 23
1.3.9 Висновки 24
1.4 Порівняльний аналіз інструментів автоматизованого тестування мобільних застосунків 25
1.5 Характеристика фреймворку Appium 27
1.5.1 Архітектура Appium 28
1.5.2 Appium і iOS 31
1.5.3 Appium і Android 32
1.5.4 Selenium JSON wire protocol 32
1.5.5 Сесії Appium 34
1.5.6 Desired capabilities 34
1.5.7 Appium server і клієнтські бібліотеки 36
1.6 Типи існуючих мобільних за стосунків 36
1.6.1 Нативні застосунки 36
1.6.2 Веб застосунки 37

1.6.3 Гібридні застосунки 37
РОЗДІЛ 2 ДОСЛІДЖЕННЯ ФРЕЙМВОРКУ ТА ІНСТРУМЕНТАРІЮ APPIUM 38
2.1 Робота з фреймворком Appium 38
2.1.1 Розгортання фреймворку Appium 38
2.1.2 Налаштування Android SDK 39
2.1.3 Налаштування Genymotion Android Emulator 41
2.1.4 Підключення реальних мобільних пристроїв 44
2.1.5 Пошук локаторів елементів інтерфейсу у Appium Inspector 46
РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОДУКТУ 55
3.1 Програмна реалізація тестування веб-застосунку 55
3.2 Програмна реалізація тестування гібридного застосунку 79
3.3 Програмна реалізація тестування нативного застосунку 83
РОЗДІЛ 4 РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ 89
4.1 Чому все ж таки Appium 89
4.2 Аналіз результатів дослідження 90
ВИСНОВКИ 91
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 92

ВСТУП

Актуальність теми

Останнім часом в області інформаційних технологій в зв'язку з постійно зростаючою конкуренцією особливої актуальності набувають питання тестування, як діяльності, що підвищує якість програмних продуктів. На сьогоднішній день одним з необхідних видів тестування, що застосовуються в ході процесу розробки і модифікації програмних продуктів, є, так зване, регресійне тестування — повторне тестування модернізованих частин програми. Однак, в умовах проектів промислового масштабу тестування в цілому, і регресійне тестування зокрема, є дуже дорогою і трудомісткою діяльністю. Автоматизація процесу тестування дозволяє за допомогою спеціальних технологій та інструментів один раз розробити тести та виконувати їх при кожній зміні продукту автоматично.

На сьогоднішній день засоби автоматизації для модульного ре інтеграційного тестування вже визначились і досить широко використовуються при розробці програмного забезпечення [4]. Натомість засоби системного (функціонального) тестування досі розвиваються, особливо в області тестування мобільних застосунків. Це зумовлено більшим діапазоном значень характеристик мобільних пристроїв, обов’язковістю кросплатформової роботи сучасних мобільних застосунків, наявністю на ринку мобільних застосунків різних типів: наітивних, веб та гібридних. Тому дослідження сучасних технологій та інструментів автоматизованого тестування мобільних застосунків є актуальним.

Мета і завдання дослідження

Метою роботи є вивчення засобів тестування мобільних застосунків фреймворку Appium, особливостей організації тестування наітивних, веб та гібридних мобільних застосунків, розробка оптимізованої методики створення тестів із використанням інструментів Selenium.

З точки зору інженерії програмного забезпечення досягнення поставленої мети передбачає:

- розгортання середовища підтримки роботи фреймворку: Java Development Kit, Android Software Development Kit, Node.js, Android-емулятора Genymotion та їх налаштування;
- розгортання фреймворку Appium та його інструментів: Appium Server GUI, Appium Inspector, консольної версії appium, Appium-doctor;
- налаштування інтегрованого середовища розробки IntelliJ IDEA та створення проекту на Java з автоматичним побудуванням за допомогою Gradle.
- підключення до проекту необхідних залежностей, зокрема, тестового фрймворку JUnit (5 версії) та фреймворку Appium;
- визначення наборів Desired Capabilities для підключення наітивних, веб та гібридних додатків;
- визначення оптимальних інструментів для пошуку локаторів елементів інтерфейсу наітивних, веб та гібридних додатків;
- розробка тест-плану для тестування мобільної версії працюючого Інтернет-магазину з метою імплементації тестів, що дозволять дослідити особливості використання технологіх Appium та її засобів;
- проектування та розробку тест-кейсів на основі розробленого тест-плану;
- визначення оптимальних засобів розробки тестів.

Достовірність отриманих результатів підтверджується практичним застосуванням тестів до мобільної версії працюючого Інтернет-магазину, а також публікаціями по темі магістерської роботи.

Об'єкт дослідження

Об'єктом дослідження є фреймворк для автоматизованого тестування мобільних застосунків Appium.

Предмет дослідження

Предметом дослідження є аспекти оптимізації технології розробки тестів мобільних застосунків.

Методи дослідження

Для розв'язання поставлених завдань використовуються наступні методи:

- розгляд досліджень в області автоматизованого тестування мобільних застосунків;
- порівняння існуючих рішень та проблем, які вони вирішують;
- аналіз найбільш перспективних технологій;
- застосування перспективних технологій для тестування реальних застосунків для підтвердження їх ефективності.

Наукова новізна одержаних результатів

Наукову новизну роботи представляють наступні результати:

Розроблено методику використання інструменту Selenium IDE для пошуку локаторів веб-елементів у веб та гібридних мобільних застосунках з подальшим експортом отриманого тестового скрипту у Junit-код з командами Selenium WebDriver, яка є більш ефективною, аніж використання інструменту Appium Inspector.

Практичне значення одержаних результатів

Використання Selenium IDE для пошуку локаторів веб-елементів з подальшим експортом отриманого тестового скрипту у Junit-код дозволяє:

- автоматично визначати локатори веб-елементів шляхом створення тестових сценаріїв у Selenium IDE;
- підвищити швидість розробки тестів;
- розширити функціонал, який буде тестуватись і тим самим покращити тестове покриття коду програмного продукту.

Апробація результатів

1. Ревякін Д.С. магістрант, Коломоєць Г.П. доцент, канд. фіз.-мат. наук — науковий керівник. Дослідження технології тестування мобільних застосунків Appium. Збірник наукових праць студентів, аспірантів і молодих вчених «Молода наука-2021» / Запорізький національний університет. Запоріжжя : ЗНУ, 2021. Т.5. С. 94-96.
2. Ревякін Д.С. магістрант, Коломоєць Г.П. доцент, канд. фіз.-мат. наук — науковий керівник. Appium як сучасний інструмент автоматизації тестування мобільних застосунків. Матеріали I Всеукраїнської науково-практичної конференції здобувачів вищої освіти, аспірантів та молодих вчених «Актуальні питання сталого науково-технічного та соціально-економічного розвитку регіонів України». Запорізький національний університет. Запоріжжя : ЗНУ, 2021. 527с. Запоріжжя: Запорізький національний університет, 2021. С.340-341.

Глосарій

Desired Capabilities — це ключі та значення, закодовані в об’єкті JSON, які надсилаються клієнтами Appium на сервер, коли запитується новий сеанс автоматизації. Вони розповідають водіям Appium всі важливі речі про те, як ви хочете, щоб ваш тест працював. Кожен клієнт Appium створює можливості специфічним для мови клієнта, але в кінці дня вони надсилаються в Appium як об’єкти JSON [3].

Java Development Kit (скорочено JDK) — безкоштовно поширюваний компанією Oracle Corporation (раніше Sun Microsystems) комплект розробника додатків на мові Java, що включає в себе компілятор Java (javac), стандартні

бібліотеки класів Java, приклади, документацію, різні утиліти і виконавчу систему Java Runtime Environment (JRE).

JUnit — фреймворк, розроблений для тестування програм, написаних з використанням технології Java. Він входить в сімейство фрейморков для тестування xUnit.

Node або *Node.js* — програмна платформа, заснована на движку V8 (здійснює трансляцію JavaScript в машинний код), що перетворює JavaScript з вузькоспеціалізованого мови в мову загального призначення. Node.js додає можливість JavaScript взаємодіяти з пристроями введення-виведення через свій API, написаний на C++, підключати інші зовнішні бібліотеки, написані на різних мовах, забезпечуючи виклики до них з JavaScript-коду. В основі Node.js лежить подієво-орієнтоване і асинхронне (або реактивне) програмування з неблокуючим введенням/виведенням.

Selenium WebDriver — це інструмент для автоматизації дій веб—браузера. У більшості випадків використовується для тестування Web-додатків, але цим не обмежується. Зокрема, він може бути використаний для виконання рутинних завдань адміністрування сайту або регулярного отримання даних з різних джерел (сайтів).

Appium — це інструмент автоматизації з відкритим вихідним кодом для запуску сценаріїв і тестування власних додатків, мобільних веб-додатків і гібридних додатків на Android або iOS за допомогою веб-драйвера. До складу Appium входить Appium Server GUI, Appium Inspector, консольна версія appium, appium-doctor.

Автоматизоване тестування програмного забезпечення — це метод тестування програмного забезпечення, який використовує програмні засоби для запуску та контролю виконання тестів.

Звіт про помилки/дефекти (Bug Report) — це документ, що описує ситуацію або послідовність дій, що призвели до некоректної роботи об'єкта тестування, із зазначенням причин і очікуваного результату.

Валідація (Validation) — це процес перевірки того, що вимоги конкретного користувача продукту, послуг або системи задоволені.

Верифікація (Verification) — це процес перевірки узгодження різних артефактів проекта: правил, стандартів, специфікації, програмного коду тощо між собою.

Тест дизайн (Test Design) — це етап процесу тестування програмного забезпечення, під час якого проектуються і створюються тестові випадки (тест кейси) відповідно до визначених раніше критеріїв якості та цілей тестування.

Тестовий випадок (Test Case) — це артефакт, що описує сукупність кроків, конкретних умов, вхідних параметрів та очікуваних результатів, необхідних для перевірки реалізації функції, що тестується, або її частини.

Тестування програмного забезпечення (Software Testing) — перевірка відповідності між реальною і очікуваною поведінкою програми, що здійснюється на кінцевому наборі тестів, обраному певним чином. [2] У більш широкому сенсі, тестування — це одна з технік контролю якості, що включає в себе активності з планування робіт (Test Management), проектування тестів (Test Design), виконання тестування (Test Execution) і аналізу отриманих результатів (Test Analysis).

Тестовий план (Test Plan) — це документ, що описує весь обсяг робіт з тестування, починаючи з опису об’єкта, стратегії, розгляду, критеріїв початку та завершення тестування, необхідного в процесі роботи обладнання, спеціальних знань, а також оцінки ризиків з варіантами їх вирішення.

РОЗДІЛ 1 ІНСТРУМЕНТИ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ МОБІЛЬНИХ ЗАСТОСУНКІВ

1.1 Історія розвитку тестування програмного забезпечення

Перші програмні системи розроблялися у межах програм наукових досліджень чи програм потреб міністерств оборони. Тестування таких продуктів проводилося строго формалізоване із записом всіх тестових процедур, тестових даних, отриманих результатів. Тестування виділялося в окремий процес, який починався після завершення кодування, але при цьому зазвичай виконувалося тим же персоналом.

У 1960-х багато уваги приділялося «вичерпному» тестуванню, яке має проводитись з використанням усіх шляхів у коді або всіх можливих вхідних даних. Було зазначено, що в цих умовах повне тестування ПЗ неможливе, тому що, по-перше, кількість можливих вхідних даних дуже велика, по-друге, існує безліч шляхів, по-третє, складно знайти проблеми в архітектурі та специфікаціях. З цих причин «вичерпне» тестування було відхилено та визнано теоретично неможливим.

На початку 1970-х тестування ПО позначалося як «процес, спрямований демонстрацію коректності продукту» чи як «діяльність з підтвердження правильності роботи ПО». У програмній інженерії, що зароджувалася, верифікація ПЗ значилася як «доказ правильності». Хоча концепція була теоретично перспективною, на практиці вона вимагала багато часу і була недостатньо всеосяжною. Вирішили, що доказ правильності — неефективний метод тестування ПЗ. Однак, у деяких випадках демонстрація правильної роботи використовується і в наші дні, наприклад, приймально-здатні випробування. У другій половині 1970-х тестування представлялося як виконання програми із наміром знайти помилки, а не довести, що вона працює. Успішний тест це тест, який виявляє раніше невідомі проблеми. Цей підхід прямо протилежний попередньому. Зазначені два визначення є «парадоксом тестування», в основі якого

лежать два протилежні твердження: з одного боку, тестування дозволяє переконатися, що продукт працює добре, а з іншого — виявляє помилки в ПЗ, показуючи, що продукт не працює. Друга мета тестування є продуктивнішою з погляду поліпшення якості, оскільки дозволяє ігнорувати недоліки ПЗ.

У 1980-х тестування розширилося таким поняттям як попередження дефектів. Проектування тестів — найефективніший із відомих методів попередження помилок. У цей час стали висловлюватися думки, що необхідна методологія тестування, зокрема, що тестування має включати перевірки протягом циклу розробки, і це має бути керований процес. У ході тестування треба перевірити не лише зібрану програму, а й вимоги, код, архітектуру, тести. «Традиційне» тестування, що існувало до початку 1980-х, стосувалося тільки скомпільованої, готової системи (зараз це зазвичай називається системне тестування), але надалі тестувальники стали залучатися до всіх аспектів життєвого циклу розробки. Це дозволяло раніше знаходити проблеми у вимогах та архітектурі і тим самим скорочувати терміни та бюджет розробки. У 1980-х з'явилися перші інструменти для автоматизованого тестування. Передбачалося, що комп'ютер зможе виконати більше тестів, ніж людина, і це зробить надійніше. Спочатку ці інструменти були вкрай простими і не мали можливості написання сценаріїв скриптовими мовами.

На початку 1990-х до поняття «тестування» стали включати планування, проектування, створення, підтримку та виконання тестів та тестових оточень, і це означало перехід від тестування до забезпечення якості, що охоплює весь цикл розробки ПЗ. У цей час починають з'являтися різні програмні інструменти для підтримки процесу тестування: більш просунуті середовища для автоматизації з можливістю створення скриптів та генерації звітів, системи управління тестами, програмного забезпечення для проведення навантажувального тестування. У середині 1990-х з розвитком Інтернету та розробкою великої кількості веб-додатків особливу популярність стало набувати «гнучке тестування» (за аналогією з гнучкими методологіями програмування).

У 2000-х з'явилося ще ширше визначення тестування, коли до нього було додано поняття «оптимізація бізнес-технологій» (en:business technology optimization, BTO). BTO спрямовує розвиток інформаційних технологій відповідно до цілей бізнесу. Основний підхід полягає в оцінці та максимізації значущості всіх етапів життєвого циклу розробки ПЗ для досягнення необхідного рівня якості, продуктивності, доступності [5].

1.2 Поняття автоматизованого тестування

Автоматичне тестування — це метод тестування програмного забезпечення, який використовує засоби автоматизації для контролю виконання тестів замість тестувальника. Потім він порівнює фактичні результати з прогнозованими або очікуваними результатами. Автоматичне тестування забезпечує більшу ефективність і швидший вихід на ринок для ваших проектів [6].

Будь-яка більш-менш значуща модернізація програмного коду вимагає повторної перевірки його функціонування. Застосовувані для цього випробування часто є типовими і можуть багаторазово повторюватися в послідовних циклах модернізації програми. Автоматизоване тестування скорочує етап тестування і вивільняє головний ресурс компанії — робочий час фахівців. Одночасно, автоматизоване тестування виключає помилки, які виникаюь внаслідок "людського фактору", що підвищує якість продукту.

До теперішнього часу більшість компаній все ще практикують тестування без використання засобів автоматизації. Іноді для цього формується команда фахівців. Але ще частіше тестуванням займаються самі розробники. Вони намагаються виконати всілякі дії користувачів і перевіряють, чи досягнуто в програмі очікуваний результат. Така практика, яку ми називаємо "ручним тестуванням", не дозволяє комплексно перевіряти багатофункціональні системи за відведені проектом терміни, що призводить до різного роду негативних наслідків [7].

Технологічно автоматизація дозволяє використовувати одні і ті ж тести багаторазово. Тести виконуються відносно швидко, що дозволяє знизити витрати (вивільняється час, який раніше витрачався на ручне тестування).

Для комплексного тестування необхідно створити значну кількість наборів цих даних. При "ручному тестуванні" обмежуються вихідні дані, виходячи з деяких припущень, для скорочення часу тестування. При автоматизованому тестуванні немає необхідності обмежувати набори вихідних даних, що дозволяє охопити набагато більше функцій продукту та підвищити тестове покриття коду програмного продукту.

Автоматизація тестування дозволяє підвищити надійність програмного забезпечення і знизити ризик виявлення дефектів на стадії промислової експлуатації. Підвищується точність тестування і можливість знаходити більше дефектів на ранніх етапах. Стає можливим виявляти й усувати вузькі місця продуктивності системи протягом усього життєвого циклу розробки. За допомогою автоматизації можна побачити точну картину продуктивності системи на всіх рівнях, включаючи введення в промислову експлуатацію.

При виборі технологій та засобів для автоматизованого тестування необхідно проаналізувати такі фактори:

- частку ринку в секторі засобів автоматизованого тестування програмного забезпечення;
- підтримку кросплатформовості для інструменту;
- можливість тестуваня кросплатформових застосунків;
- можливість аналізувати вихідний код інструменту;
- якість технічної підтримки, наявність консультаційної та тренінгової підтримки;
- наявність та величина комьюніті;
- вартість ліцензій і гнучкість в ліцензуванні, оцінка повернення інвестицій.

Також важливим фактом є те, що автоматизовані тести забезпечують ранній та частий відгук, це зумовлено таким чином, що після того, як автоматизований тест деякої частини функціональності пройшов успішно, він і проходитиме успішно доти, поки ця функціональність не буде навмисно змінена. Коли ми плануємо зміни в додатку, паралельно змінюємо тести, щоб вони відповідали змінам основного коду. Якщо автоматизований тест зненацька дає збій, регресивний дефект може бути викликаний зміною коду. Запуск комплекту автоматизованих тестів при кожній фіксації нового коду в системі керування версіями допомагає гарантувати швидке перехоплення регресивних помилок. Швидкий відгук означає, що зміна залишається свіжою у свідомості деякого програміста, і тому може швидко усунути несправність, ніж у тому випадку, коли вона виявляється набагато пізніше. Швидке виявлення помилок означає, що їх дешевше виправляти.

Автоматизовані тести, що запускаються регулярно, часто є детектором змін. Вони дають команді можливість дізнатися про зміни, що відбулися з моменту останнього успішного збирання. Наприклад, чи не було якихось негативних побічних ефектів із останньою збіркою? Якщо автоматизований комплект тестів забезпечує хороше покриття, то він може легко виявити глибокі дефекти, які ніколи не виявлять ручні тестувальники.

Якщо регресивні тести не автоматизовані, то, швидше за все, вони не проганятимуться при кожній ітерації або щодня. І тоді проблема негайно виникає в кінці гри, коли команді доводиться виконувати всі регресивні тести.

Помилки, які можна було б виявити рано, виявляються пізно, і багато переваг раннього тестування губляться.

1.3 Бар’єри автоматизованого тестування

У автоматизації тестування бувають і так звані бар’єри.

1.3.1 Список Брета Петтіхорда

Ще у 2001 р. Брет Петтіхорд (Bret Pettichord) перерахував кілька проблем, від яких страждає автоматизація. Вони досі актуальні, але переважно стосуються команд, які не включають автоматизацію у свій процес розробки.

Список Брета демонструє які проблеми можна отримати, якщо не зробити автоматизацію невід'ємною частиною процесу роботи над проектом.

- Виділення часу на автоматизацію тестів за залишковим принципом не дозволяє зосередитись на ній належним чином.
- Нестача ясності цілей.
- Настача досвіду.
- Це велика реорганізація, тому що ви втрачаєте досвід, який могли мати.
- Реакція на безвихідь часто є причиною вибору на користь автоматизації і в цьому випадку вона буде скоріше бажанням, ніж реалістичним планом.
- Можливо, вам просто лінькі думати про тестування; задоволення пов’язане з автоматизацією, а не з тестуванням.
- Зосередження на вирішенні технічної проблеми може призвести до втрати бачення того, що результат відповідає потребам тестування [8].

1.3.2 Позиція програмістів «Навіщо автоматизувати»

Програмісти, які звикли працювати у традиційному середовищі, де є окрема, невидима команда забезпечення якості (QA), що займається тестуванням, можуть взагалі не замислюватись про автоматизацію функціональних тестів. Деякі програмісти не надто замислюються про тести, тому що для них "страхованою сіткою" служить команда QA, яка відловлює помилки до виходу випуску. Тривалі цикли каскадної розробки ще більше віддаляють тестування програмістів. На момент, коли невидимі тестувальники приступають до своєї роботи, програмісти вже переходять до наступного випуску. Дефекти став-

ляться в чергу для подальшого виправлення, і ніхто не відповідає за них. Навіть програмісти, які застосовують керовану тестами розробку та застосовують автоматизацію тестування на рівні модулів, можуть не думати про те, як виконувати приймальні тести, що виходять за межі рівня модулів.

Ключем до переконання програмістів та інших членів команди на користь та важливості автоматизації є освіта [8].

1.3.3 Початкові інвестиції

Навіть коли вся команда працює над проблемою, автоматизація вимагає серйозних інвестицій, які скоро окупаються. Потрібен час та дослідження, щоб вибрати підходящий каркас тестування та вирішити, чи варто його будувати самостійно або скористатися готовим інструментом. Також може знадобитися нове апаратне та програмне забезпечення. Членам команди може знадобитися багато часу, щоб навчитися використовувати оснащення автоматизованого тестування.

Багато людей стикалися з тим, що зусилля щодо впровадження автоматизації не виправдовували себе. Їхні організації купували готові інструменти зйомки-відтворення, передавали їх команді QA та очікували, що вони вирішать усі проблеми автоматизації тестування. Часто такі інструменти просто залишалися припадати пилом на полиці. У компанії могли існувати тисячі рядків сгенерованих сценаріїв тестування графічного інтерфейсу користувача, але жодної живої душі, яка знала, що вони роблять; аналогічно були справи з тестовими сценаріями, які неможливо підтримувати і які стали марними.

Кваліфікація проектувальника тестів дуже впливає на віддачу від автоматизації. Погані прийоми породжують тести, які важко зрозуміти і підтримувати, і які можуть видавати результати, що важко інтерпретуються, або давати помилкові збої, на дослідження яких може знадобитися додатковий час.

Команди з неадекватним рівнем підготовки та кваліфікацією можуть вирішити, що спроби отримати віддачу від інвестицій у тестування не варті їхнього часу.

Правильні прийоми проектування тестів породжують прості, добре спроектовані, що піддаються постійному рефакторингу і тести, що легко супроводжуються.

Згодом з'являються бібліотеки тестових модулів та об'єктів, які полегшують автоматизацію нових тестів.

Нам відомо, що часто зафіксувати показники нелегко. Наприклад, зафіксувати час, необхідний написання і супровід автоматизованих тестів, порівняно із запуском тих самих регресивних тестів вручну практично неможливо.

Також непросто зафіксувати, скільки коштує усунення дефектів за кілька хвилин після їх появи, порівняно з вартістю знаходження та виправлення проблем після закінчення ітерації. Багато команд навіть не намагаються відслідковувати таку інформацію. Але не маючи можливості наочно продемонструвати економічний ефект від запровадження автоматизації тестування, важко переконати менеджмент у доцільності інвестицій у цю діяльність. Нестача показників, що демонструють віддачу від інвестицій, ускладнює завдання зміни старих звичок команди [8].

1.3.4 Плинність коду

Автоматизація тестів інтерфейсу користувача — складне завдання, оскільки дані інтерфейси мають тенденцію часто змінюватися в процесі розробки. Це одна з причин, через які простий запис та відтворення дій користувача рідко застосовуються в гнучких проектах.

Якщо команда прагне розробляти хороший дизайн на базі бізнес-логіки і доступу до бази даних, що лежить в основі, і серйозні зміни відбуваються часто, то важко підтримувати навіть автоматизацію тестів, які перевіряють код, що знаходиться нижче графічного інтерфейсу користувача — на рівні API. Якщо при проектуванні системи було мало уваги можливості тестування, знайти спосіб автоматизації тестів може виявитися важко і дорого. Для створення програми, що тестується, програмісти і тестувальники повинні працювати разом.

Хоча дійсний код і реалізація, подібна до графічного інтерфейсу користувача, мають тенденцію часто змінюватися в гнучкій розробці, призначення коду змінюється рідко. Організуйте тестування коду на основі призначення програми, а не на основі його реалізації; це дозволить не відставати від розробки [8].

1.3.5 Успадкований код

Автоматизацію впровадити набагато легше, якщо ви пишете новий код в архітектурі, спроектованій з урахуванням можливості тестування. Написання тестів для існуючого коду, який має мінімум тестів або не має їх зовсім — завдання, щонайменше, лякає. Це здається практично неможливим для команди новачків у гнучкій розробці та автоматизації тестування.

Іноді потрібно автоматизувати тести, щоб мати можливість виконати рефакторинг деякого успадкованого коду, але цей код не спроектований з урахуванням тестованості, тому автоматизувати тести важко навіть на рівні модулів [8].

1.3.6 Страх

Завдання автоматизації тестів лякає тих, хто не має відповідних навичок, а іноді навіть тих, хто має. Програмісти можуть писати продуктивний код, але не мати досвіду в написанні автоматизованих тестів. Тестувальники можуть мати багатого досвіду програмування, і впевнені у здібностях автоматизації тестів.

Тестувальники — непрограмісти часто чують, що їм нічого робити у світі гнучкої розробки. Ми думаємо інакше. Жоден індивідуальний тестувальник не повинен турбуватися про те, як здійснити автоматизацію. Це проблема команди в цілому, і зазвичай у команді завжди є безліч програмістів, які можуть у цьому допомогти. Секрет успіху — у готовності опановувати нові ідеї. Витратьте на це бодай день [8].

1.3.7 Старі звички

Коли ітерація йде не гладко, і команда не встигає завершити всі завдання програмування та тестування до кінця ітерації, члени команди можуть запанікувати.

Було підмічено, що коли люди впадають у паніку, вони повертаються до старих звичок, навіть незважаючи на те, що це не може забезпечити добрий результат.

Отже, можна сказати: “Ми маємо термін здачі 1 лютого. Якщо ми хочемо вкластися вчасно, то не можемо витрачати час на автоматизацію тестів. Ми повинні зробити все можливе, щоб протестувати вручну все, що встигнемо, та сподіватися на краще. Ми завжди зможемо автоматизувати випробування пізніше”.

Це — згубний шлях. Деякі ручні випробування можуть бути виконані, але, можливо, не залишиться часу на важливі ручні дослідні випробування, які виявили б помилку, що стоїть компанії сотень тисяч доларів через втрачені продажі. Потім, оскільки ми не закінчили автоматизовані тести, це завдання відкладається наступну ітерацію, скорочуючи обсяг корисних засобів, які ми встигнемо реалізувати. З кожною новою ітерацією ситуація продовжуватиме погіршуватися [8].

1.3.8 Чи можливо подолати бар’єри?

Гнучкий повнокомандний підхід є основою подолання проблем автоматизації.

Програмісти-новачки в гнучкій розробці зазвичай орієнтовані на поставку коду, незалежно від того, сповнений він помилок чи ні, якщо вони укладаються у строки.

Керована тестами технологія більше орієнтована на проектування, ніж на тестування, тому бізнес-орієнтовані тести можуть бути втрачені з уваги.

Необхідний сильний лідер і відданість команди якості, щоб змусити всіх думати про те, як писати, використовувати та запускати технологічно-орієнтовані та бізнес-орієнтовані тести. Залучення всієї команди в автоматизацію тестування може стати проблемою культури [8].

1.3.9 Висновки

- Автоматизація необхідна для створення "страхувальної сітки", отримання суттєвого відгуку, зведення до мінімуму рівня технічного боргу та допомоги в управлінні кодуванням.
- Страх, недолік знань, негативний минулий досвід впровадження автоматизації, код, що змінюється, успадкований код — все це поширені бар'єри на шляху автоматизації.
- Автоматизація регресивних тестів, запуск їх у складі автоматизованого процесу складання та усунення причин дефектів скорочує технічний обов'язок і дозволяє розробляти надійний код.
- Автоматизація регресивних тестів та стомлюючої ручної роботи звільняє команду для вирішення важливіших завдань, таких як дослідження.
- Команди, що практикують автоматизоване тестування та автоматизований процес складання, демонструють стабільнішу продуктивність.
- Без автоматизації регресивних тестів обсяг ручного регресивного тестування продовжуватиме зростати в обсязі і, зрештою, може просто ігноруватися.
- Культура та історія команди можуть ускладнити програмістам віддати пріоритет автоматизації бізнес-орієнтованих тестів перед кодуванням нових коштів. Використання гнучких принципів та цінностей допомагає всій команді подолати бар'єри, що стоять на шляху автоматизації тестування [8].

1.4 Порівняльний аналіз інструментів автоматизованого тестування мобільних застосунків

Аналіз джерел [9] показує, що у 2021 році найбільш популярними фреймворками для автоматизованого тестування мобільних застосунків є:

- Appium.
- Robotium.
- TestComplete Mobile.
- Espresso.
- Kobiton.
- Selendroid.

В таблиці 1 представлено порівняльний аналіз цих інструментів для автоматизованого тестування мобільних застосунків:

Таблиця *1*

Порівняльний аналіз інструментів для автоматизованого тестування мобільних застосунків

Параметр	Appium	Robotium	TestComplete Mobile	Espresso	Kobiton	Selendroid
Розробник	Sauce Labs	Renas Veda	SmartBear Software	Google	Kobiton, Inc.	Selendroid
Тестування наітивних/веб/гібридних застосунків	Так/Так/Так	Так/Ні/Так	Так/Так/Так	Так/Ні/Так	Так/Так/Так	Так/Так/Так
Мови програмування	Java/Objective-C/JS+node.js/PHP/Ruby/Python/C#	Java	JavaScript/Python/VBScript/JScript/DelphiScript	Java/Kotlin	Java/Objective-C/JS+node.js/PHP/Ruby/Python/C#	Java/Objective-C/JS+node.js/PHP/Ruby/Python/C#
Мобільні операційні системи	Android + iOS	Android	Android + iOS	Android	Android + iOS	Android
Ціна	Open-Source	Open-Source	9,6$/month	Free	16000$/year	Open-Source
Підтримка емуляторів	Так	Підтримка віртуальних девайсів Android Studio	Так	Так	Так	Так

Результати аналізу

Якщо привести підсумки з аналізу інструментів для автоматизації мобільних застосунків, то можно підвести, чому саме Appium. Бо саме у цьому фреймоврку просто взяли і реалізували протокол Selenium для роботи з мобільними пристроями. Нічого зайвого — тільки управління девайсом. А це означає що:

- Пиши на будь-якій мові програмування, яка необхідна користувачеві.
- Використовуй будь-який тестовий фреймворк.
- Підключайся до Selenium-Grid і виконуй тести параллельно.
- Не потрібно сплачувати підписку для тестування застосунків, тому що Appium є Open-Source фреймворком.

1.5 Характеристика фреймворку Appium

Appium був розроблений для задоволення потреб мобільної автоматизації відповідно до філософії [13, 14], викладеної такими чотирма прицнипами:

- Вам не доведеться перекомпілювати застосунок або будь-яким чином змінювати його, щоб його автоматизувати, таким чином у архітектурі Appium використовуються вендорні фреймворки, такі як: XCUITest, UIAutomation для iOS та UIAutomator2 для Android, тому нам не доведеться компілювати «Appium Specific», або «Third-party» код.
- Ви не повинні бути заблоковані в певній мові чи фреймворку, щоб писати та запускати свої тести. Цей пункт був реалізований за рахунок обгортки вендорних фреймворків у єдиний API, що називається WebDriver API. WebDriver (він же «Selenium WebDriver») визначає клієнт-серверний протокол (відомий як JSON Wire Protocol). Враховуючи цю клієнт-серверну архітектуру, клієнт, написаний будь-якою мовою, може використовуватися для відправки відповідних запитів HTTP на сервер. Вже є клієнти, написані всіма популяр-

ними мовами програмування. Це також означає, що ви можете використовувати будь-який тест-раннер і тестовий фреймворк, який забажаєте; клієнтські бібліотеки — це просто клієнти HTTP, і їх можна помістити до вашого коду будь-яким способом. Іншими словами, клієнти Appium і WebDriver технічно не є «тестовими фреймворками» — це «бібліотеки автоматизації». Ви можете керувати своїм тестовим середовищем як завгодно!

- Фреймворк мобільної автоматизації не повинен заново винаходити колесо, коли справа доходить до API автоматизації. Цей пункт був реалізовний у тому ж напрямку, як і пункт №2, тобто WebDriver став де-факто стандартом для автоматизації веб-браузерів. Навіщо робити щось зовсім інше для мобільних пристроїв? Замість цього у Appium розширили протокол додатковими методами API, корисними для мобільної автоматизації.
- Фреймворк мобільної автоматизації має бути з відкритим кодом.

1.5.1 Архітектура Appium

Appium — це HTTP-сервер, написаний на Node.js, який представляє у собі вузол, який необхідний для його запуску. Якщо вузли були б написані в Python, то нам знадобвися би Python, якби вони були написані на Java, то нам би знадобилась Java. Нам потрібне правильне середовище для запуску кожного програмного застосунка, якщо воно не скомпільоване в машинний код та не збережено у вигляді власного двійкового коду для даної архітектури, що не відноситься до JS/Node (або Java, Perl, Python, Ruby і т.д.). (див. рис. 1)

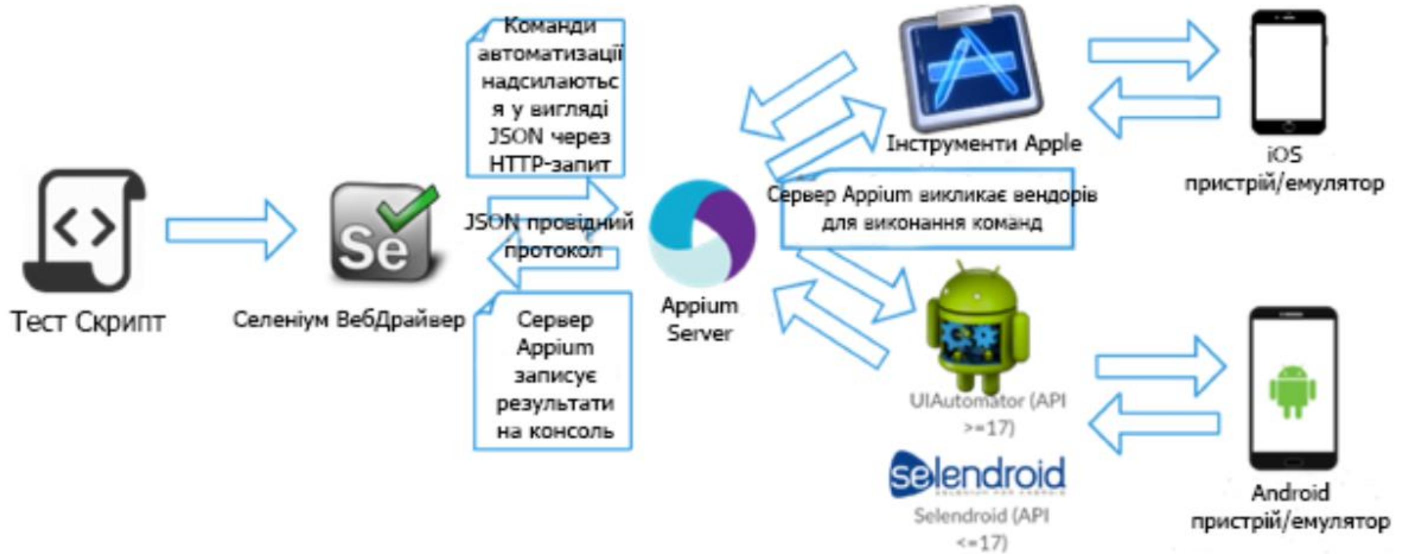


Рис. *1 Робочий процес Appium*

Якщо перейти до вихідного коду Appium, то можна побачити, який йому необхідний вузол, та якої версії у файлі package.json:

```
"engines": {
     "node": ">=4",
     "npm": ">=3"
}
```

Також Node використовуються для розпаралелювання.

WebDriver — це інтерфейс віддаленого керування, який дає змогу аналізувати та контролювати User Agents. Він забезпечує незалежний від платформи та мови провідний протокол як спосіб для позапроцесних програм віддалено інструктувати поведінку веб-браузерів.

Надається як набір інтерфейсів для виявлення та керування елементами DOM у веб-документах та керування поведінкою User Agent. Насамперед він призначений для того, щоб дозволити веб-авторам писати тести, які автоматизують User Agent з окремого процесу контролю.

Протокол WebDriver складається зі зв’язку між:

- Local End — представляє клієнтську сторону протоколу, яка зазвичай представлена у формі специфічних для мови бібліотек, що надають API поверх протоколу WebDriver.
- Remote End — розміщує серверну частину протоколу. Визначення поведінки віддаленого кінця у відповідь на протокол WebDriver становить найбільшу частину цієї специфікації.

Для Remote End стандарту визначають два широкі класи відповідності, відомі як типи вузлів:

- Intermediary node — це вузли, які діють як проксі, реалізуючи як локальний, так і віддалений кінець протоколу. Однак не очікується, що вони безпосередньо реалізують віддалені кінцеві кроки. Кажуть, що всі вузли між конкретним проміжним вузлом і локальним кінцем знаходяться нижче по потоку від цього вузла. І навпаки, будь-які вузли між конкретним проміжним вузлом і вузлом кінцевої точки називаються вихідними.
- Endpoint node — це віддалений кінець ланцюга вузлів, який не є проміжним вузлом. Вузол кінцевої точки реалізується user agent або подібною програмою.

Основні команди Appium/Selenium по категоріям: відкриття сайту, пошук елементів, виконання дій, перевірка наявності та відповідні команди (ендпоінти) WebDriver:

- Для того щоб відкрити сайт, використовується команда `driver.get(“url”);` ця команда говорить веб-драйверу про необхідність відкрити потрібну користувачеві сторінку в інтернеті.
- Пошук елементів виконується за рахунок локаторів, якими названі кожні елементи на екрані. Для того, щоб знайти необхідний елемент, використовується команда `driver.findElement` елементи можно знаходити за їх xPath, CSS Selector, id, Name, className.
- Інтерактивність з елементами описується згідно до ситуацій, яка інтерактивність безпосередньо необхідна над елементом. Наприклад якщо нам

необхідно видалити дані в полі, які там відображаються, то використовується команда

```
driver.findElement(By.локатор("локатор")).clear();
```

Якщо ж нам потрібно вписати якісь дані в поле, то використовується та ж команда, що описана вище за винятком, що замість `.clear()` ми будемо використовувати `.sendKeys();` якщо ж нам потрібно клікнути на елемент, то використовується команда `.click()` також у випадках, коли сторінку необхідно скроллити, ми не використовуємо ніяких команд, бо веб-драйвер сам знайде необхідний елемент на сторінці за вказаним локатором.

- Для того щоб виконити перевірку на наявність, а саме звірити фактичний результат з очікуваним, спочатку задається строкова змінна, яка буде зберігати в собі очікуванний результат, наприклад `String expected = "Профіль успішно збережено"` після чого ми за допомогою описаних раніше локаторів розшукуємо на сторінці необхідний нам елемент, що стане очікуваним результатом за допомогою команди `WebElement messageElement = driver.findElement("шукаємо за локатором");` і потім вказуємо асерти які виконають перевірку очікуваного результата з фактичним за допомогою команди

```
Assertions.assertEquals(expected,messageElement.getText());
```

1.5.2 Appium і iOS

На iOS-пристрої, Appium використовує XCUI Driver — це комбіноване рішення, яке дозволяє виконувати автоматичне тестування «чорного ящика» нативних додатків iOS, tvOS і веб застосунків WebKit. Нативне тестування базується на платформі Apple XCTest і форку сервера WebDriverAgent від Facebook (оригінальний проект більше не підтримується). Зв'язок із веб-переглядами здійснюється за допомогою протоколу віддаленого налагоджувача Webkit. Зв'язок реальних пристроїв забезпечується бібліотекою appium-ios-device. Зв'язок симуляторів забезпечується бібліотекою appium-ios-simulator [1] .

Виконуваний скрипт віддається HTTP-запитом Appium-сервера у вигляді JSON. Appium-сервер посилає команду інструментам (UIAutomation2). Інструменти шукають файл bootstrap.js, який Appium-сервер передав iOS-пристрою. Потім, команди, вказані у файлі bootstrap.js виконуються оточенням iOS-інструментів. Виконавши команду, клієнт відправляє серверу звіт з деталями виконання цієї команди.

Схожа архітектура працює і в зв'язці Appium-Android.

1.5.3 Appium і Android

На Android-пристрої, Appium використовує UIAutomator2, щоб автоматизувати застосунок. UIAutomator2 — це індивідуальна реалізація спільноти Appium, яка стабілізує та прискорює взаємодію з елементами інтерфейсу користувача.

UIAutomator2 підтримує Android 5.0 (API Level 20) та вище. Як взагалі він працює? Коли клієнт просить створити нову AndroidDriver сессію, клієнт передає залежності (desired capabilities) до Appium Node сервера. На основі `automationName` властивостях у desired capabilities, Appium виконує перенаправлення до відповідного модуля драйвера (за замовчуванням Appium переходить до appium-android-driver за відсутності `automationName`).

Якщо `automationName` є UIAutomator2, то Appium виконує перенаправлення до appium-uiautomator2-driver для створення нового сеансу.

Модуль драйвера UIAutomator2 створює сессію, встановлює appium uiautomator2 sever apk на девайс, запускає сервер (Netty) та ініціює його сеанс.

Модуль сервера UIAutomator2 після того, як сеанс сервера Netty ініціалізується модулем драйвера, буде прослуховувати запити на пристрої та відповідатиме відповідним чином, поки не буде викликано DELETE SESSION.

1.5.4 Selenium JSON wire protocol

JSON wire protocol (JSONWP) — механізм, створений командою розробників WebDriver. Цей протокол являє собою набір чітко визначених стандар-

тизованих кінцевих точок (endpoints), відкритих через RESTful API. Призначення WebDriver і JSONWP — автоматизувати тестування сайтів через браузер таких як Firefox driver, IE driver, Chrome driver і т.д.

Appium реалізує Mobile JSONWP — розширення Selenium JSONWP — і контролює різну поведінку мобільних пристроїв, наприклад установка / видалення програми за сесію.

Ось кілька прикладів кінцевих точок з API, які використовуються для взаємодії з мобільними додатками:

Лістинг *1 Приклади кінцевих точок з API, які використовуються для взаємодії з мобільними додатками*

- `/Session/:sessionId`
- `/Session/:sessionId/element`
- `/Session/:sessionId/elements`
- `/Session/:sessionId/element/:id/click`
- `/Session/:sessionId/source`
- `/Session/:sessionId/url`
- `/Session/:sessionId/timeouts/implicit_wait`

Appium надає клієнтські бібліотеки, схожі на бібліотеки WebDriver, щоб взаємодіяти з REST API. У цих бібліотеках функції виглядають, приблизно, так:

- `AppiumDriver.getPageSource();`

Цей метод викличе HTTP-запит, і отримає відповідь від кінцевої точки з API. Саме в цьому прикладі, ендпоінт, який обробляє метод getPageSource, виглядають так:

- `/ Session /: sessionId / source`

Драйвер виконає тестовий скрипт, який приходить в JSON-форматі від AppiumDriver сервера, щоб отримати source сторінки. Назад повернеться page source в форматі рядки. У разі не-HTML платформ (нативні додатки), Appium поверне XML-документ, що представляє ієрархію UI-елементів. Структура документа може відрізнятися, в залежності від платформи [11].

1.5.5 Сесії Appium

Сесія — середовище, в якому відбувається відправка команд певної програми; команда завжди виконується в контексті поточної сесії. Як ми бачили в попередньому розділі, клієнт використовує ідентифікатор сесії — параметр `sessionId` — до виконання самої команди. Клієнтська бібліотека шле запит серверу на створення сесії. Потім, сервер повертає `sessionId`, який використовується в наступних командах для взаємодії з тестовим додатком [11].

1.5.6 Desired capabilities

Desired capabilities (бажані можливості) — JSON-об'єкт (набір пар ключ-значення), відправлений клієнтом серверу, які описують особливості створюваної сесії. Існують загальні бажані можливості та специфічні для різних мобільних операційних систем. В таблиці 2 приведені основні загальні бажані можливості, а у таблиці 3 приведені основні Android залежності.

Таблиця *2*

Основні загальні бажані можливості Appium

Можливість	Опис	Найчастіше вживані значення
automationName	Фреймворк автоматизації, що буде використовуватись	Appium (за замовчуванням) або UiAutomator2 або Espresso для Android, або XCUITest для iOS
platformName	Операційна система мобільного пристроя	iOS або Android
platformVersion	Версія операційної системи мобільного пристроя	наприклад, 5
deviceName	Встановлює тип мобільного пристрою або емулятора	наприклад Android Emulator, Nexus 5 і так далі.

Продовження таблиці *2*

app	Абсолютний шлях до файлу або URL для скачування файлу в форматі .ipa, .apk, або .zip, який Appium спочатку встановить на мобільний пристрій. UiAutomator2 і XCUITest дозволяють не використовувати app. app несумісна з browserName	наприклад, "c:\\HelloAppium\\app\\Calculator.apk" або https://www.example.com/apps/Calculator.apk
browserName	Використовується при тестуванні веб-застосунків, визначає мобільний браузер для тестування	"Safari" для iOS та "Chrome", "Chromium" або "Browser" для Android

Таблиця *3*

Основні залежності Android

Можливість	Пояснення
appPackage	Визначає, який Java-пакет повинен бути запущений. Наприклад: com.android.calculator2 або com.android.settings caps.setCapability ("appPackage", "com.android.calculator2"); Або з використанням бібліотеки Appium: caps.setCapability (MobileCapabilityType.APP_PACKAGE, "com.android.calculator2");
appActivity	Визначає, яку Activity необхідно запустити з зазначеного пакета. Наприклад: MainActivity, .Settings, com.android.calculator2.Calculator caps.setCapability ("appActivity", "com.android.calculator2.Calculator");

1.5.7 Appium server і клієнтські бібліотеки

Оскільки взаємодія з Appium сервером базується на використанні REST API то код, за допомогою якого організується така взаємодія, може бути написаний на різних мовах, таких як Java, C #, Ruby, Python та інших. Appium надає клієнтські бібліотеки на вказаних мовах програмування, що підтримують розширення Appium для протоколу WebDriver. Ці бібліотеки обгортають стандартні клієнтські бібліотеки Selenium, щоб забезпечити всі звичайні команди Selenium, підтримувані JSON Wire Protocol, і додають додаткові команди, пов'язані з керуванням мобільними пристроями, такі як жести мультитач та орієнтація екрана.

1.6 Типи існуючих мобільних за стосунків

1.6.1 Нативні застосунки

Нативний додаток — це програма, розроблена для певного мобільного пристрою або платформи (наприклад, Android, iOS або Windows). Наприклад, програми для iPhone написані на Swift, а програми для Android написані на Java. Вбудовані програми також краще працюють і мають високий ступінь надійності, оскільки використовують базову архітектуру системи та вбудовані функції пристрою. Вбудовані програми можуть працювати як в онлайн-режимі, так і в автономному режимі. Native App пов'язані з мобільною операційною системою, для якої вона була розроблена, і, отже, не може працювати в жодній іншій операційній системі. Це робить розробку рідної програми дорогою, оскільки ту саму програму потрібно переписати для іншої операційної системи. Ці програми доступні для завантаження на мобільний телефон через відповідний магазин додатків.

1.6.2 Веб застосунки

Мобільний веб-застосунок — це програма, доступ до якої здійснюється через мобільний браузер. До нього можна легко отримати доступ через вбудовані браузери, такі як Safari на iOS і Chrome на Android. В основному вони розроблені з використанням таких технологій, як HTML5 або JavaScript, які надають можливості налаштування. Таким чином, вони в основному обслуговуються з сервера і не зберігаються в автономному режимі ніде на пристрої.

Веб-застосунки мають загальну базу коду, і до них можна отримати доступ, як і до будь-яких типових веб-програм, на будь-якому пристрої з браузерами. Для мобільних веб-застосунків адаптивний веб-дизайн є новим стандартом, оскільки вони повинні обслуговувати пристрої з різними розмірами екрана та роздільною здатністю. Мобільні веб-застосунки також можуть отримати доступ до функцій мобільного зв'язку, таких як набір номера телефону або відображення на основі місцезнаходження. Доступ до мобільних веб-програм можна отримати лише через дійсну мережу (Wifi/4G/3G/2G).

1.6.3 Гібридні застосунки

Гібридна програма складається в основному з веб-сайтів, упакованих у нативну обгортку. Вони в основному розроблені у веб-технологіях (HTML5, CSS, JavaScript), але працюють у нативному контейнері, створюючи таким чином відчуття, що це рідна програма. Гібридні програми покладаються на те, що HTML відтворюється в мобільному браузері, з обмеженням, що браузер вбудовується в програму. Такий підхід дозволяє мати єдину кодову базу для всіх мобільних операційних систем: iOS, Android і Windows. Рівень абстракції Web-to-Native надає доступ до можливостей, що стосуються пристрою, які в іншому випадку недоступні в мобільних веб-програмах. Приклади включають камеру, локальну пам'ять пристрою та акселерометр.

Гібридний додаток є найулюбленішим підходом для компаній із наявною веб-сторінкою. Ці компанії часто створюють гібридні програми як обгортку на веб-сторінці. Ці програми можна завантажити через відповідні магазини додатків.

РОЗДІЛ 2 ДОСЛІДЖЕННЯ ФРЕЙМВОРКУ ТА ІНСТРУМЕНТАРІЮ APPIUM

2.1 Робота з фреймворком Appium

2.1.1 Розгортання фреймворку Appium

Для встановлення Appium на комп'ютері з операційною системою Windows необхідно виконати наступні кроки:

- встановити Java Development Kit (JDK), завантаживши інсталяційний пакет з сайту https://www.oracle.com/java/technologies/downloads/#jdk17-windows, і прописати до нього шлях в змінній оточення `JAVA_HOME`, а також додавши до змінної оточення `PATH` елемент `%JAVA_HOME%\bin`.
- встановити Node.js , завантаживши інсталяційний пакет з сайту https://nodejs.org/uk/download/;
- встановити сервер `appium` за допомогою менеджера пакетів `npm` Node.js:

```
npm install appium;
```

- встановити утиліту `appium—doctor` за допомогою менеджера пакетів `npm` Node.js:

```
npm install —g appium—doctor;
```

- запустити з командного рядка утиліту `appium—doctor` для перевірки роботоспроможності Appium. Запускаємо `appium—doctor` з параметром `—ios` або `—android` відповідно до операційної системи, під якою працює застосунок, що буде тестуватись.

```
C:\WINDOWS\system32\cmd.exe - "node"  "C:\Users\Flydiecry\AppData\Roaming\npm\\node_modules\appium\build\lib\main.js"
Microsoft Windows [Version 10.0.19041.1288]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\Flydiecry>appium
[Appium] Welcome to Appium v1.22.0
[Appium] Appium REST http interface listener started on 0.0.0.0:4723
```

Рис. *2 Консольний інтерфейс Appium*

- встановити Appium Server GUI, завантаживши інсталяційний пакет з сайту https://github.com/appium/appium-desktop;
- встановити Android Studio, завантаживши інсталяційний пакет з сайту https://developer.android.com/studio, і прописати в змінній оточення `ANDROID_HOME` шлях до встановленого разом з Android Studio Android Software Development Kit (SDK), а також додавши до змінної оточення `PATH` елементи `%ANDROID_HOME%\tools` та `%ANDROID_HOME%\platform-tools;`
- встановити Android-емулятор Genymotion, завантаживши інсталяційний пакет з сайту https://www.genymotion.com/download/.

2.1.2 Налаштування Android SDK

Далі потрібно налаштувати Android SDK. Для цього в Android Studio командою Tools-SDK Manger потрібно запустити відповідний менеджер та обрати на вкадці *SDK Platforms* версії операційної системи Android для яких будуть створюватися тести (Рис. *3*).

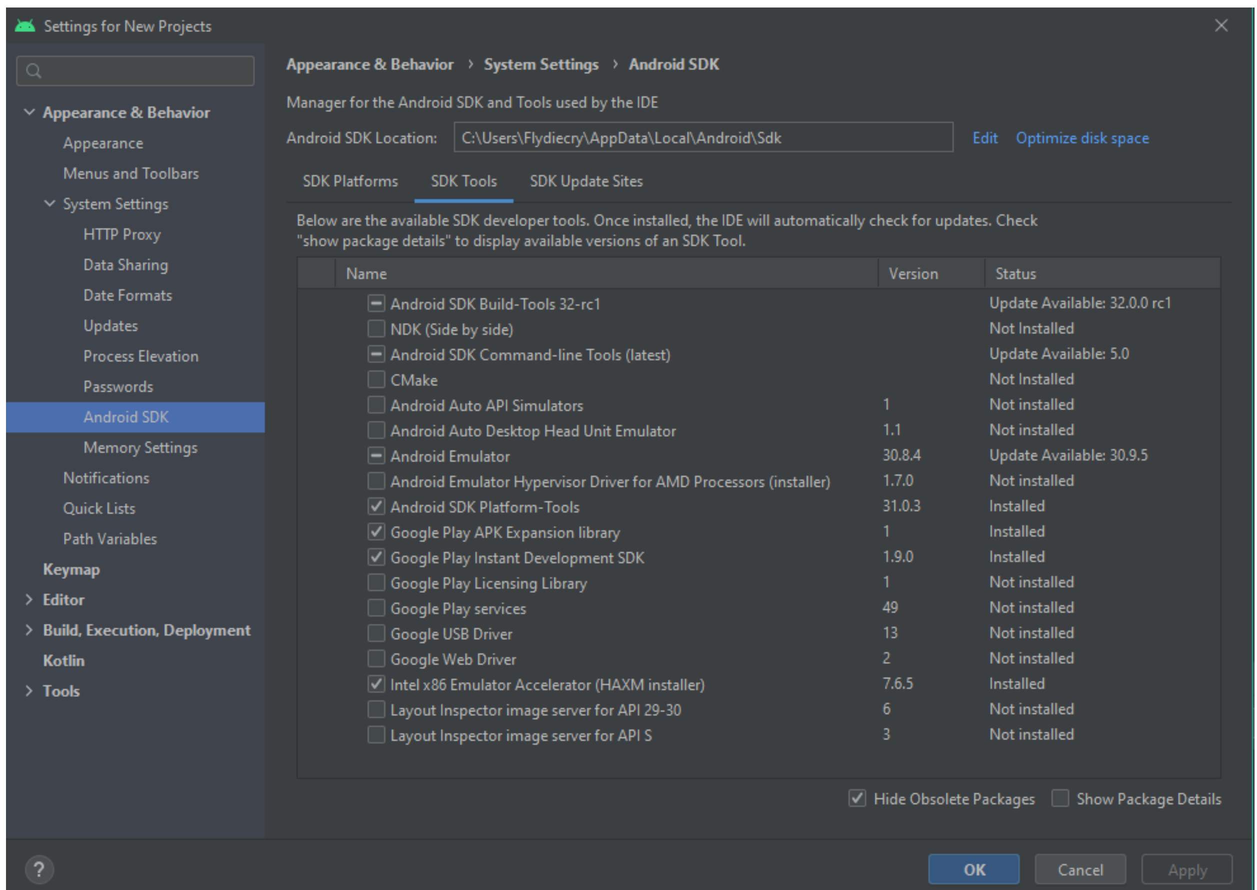


Рис. *3 Вікно SDK Platforms*

На вкладці *SDK Tools* (див. рис. 4) потрібно інсталювати Android SDK Platform-tools, Google Play APK Expansion Library, Google Play Instant Development SDK, та Intel x86 Emulator Accelerator (HAXM Installer).

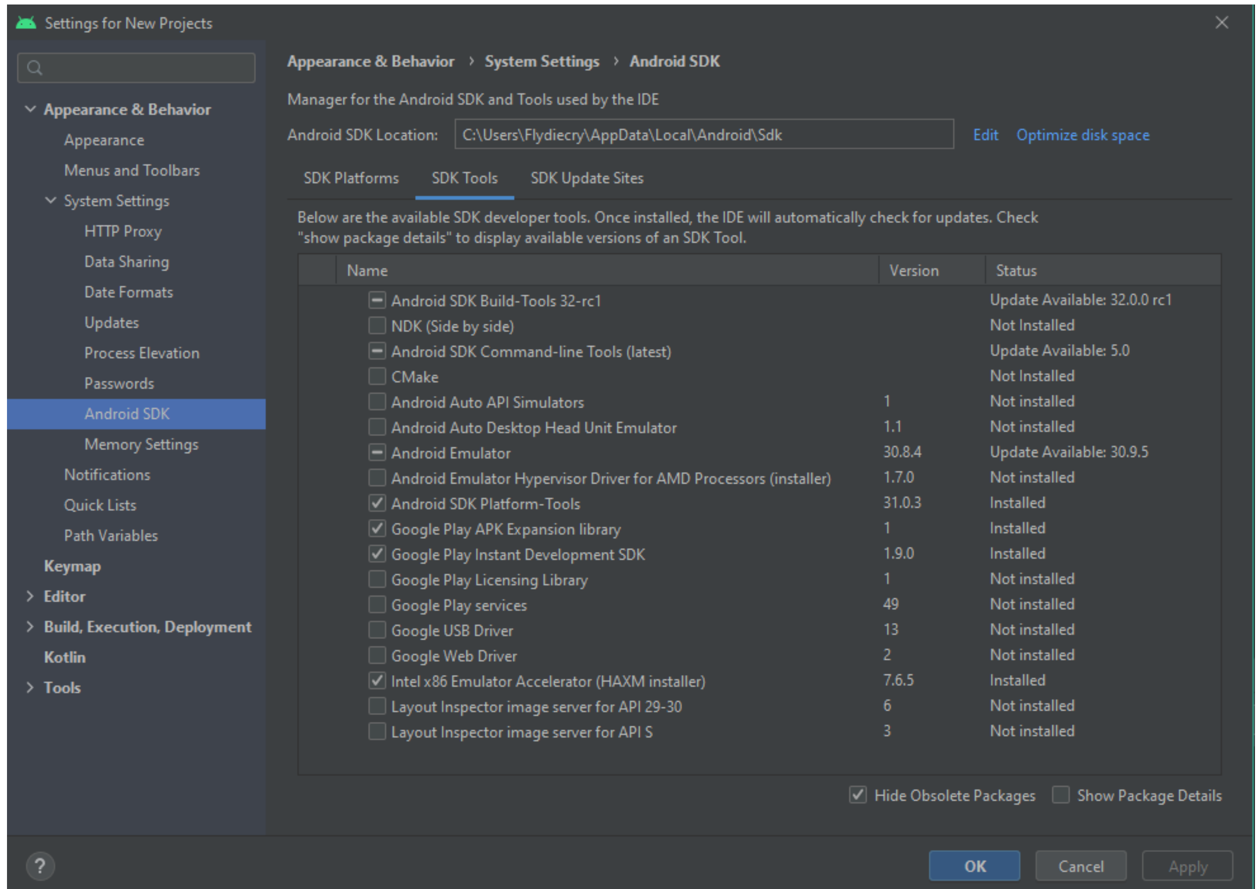


Рис. *4 Вікно SDK Tools*

2.1.3 Налаштування Genymotion Android Emulator

Емулятор – це програма, яка емулює справжній мобільний пристрій, що дозволяє вам прототипувати програму, що розробляється, або дозволяє випробувати програму, фактично не купуючи фізичний пристрій. Коли ми встановлюємо Android SDK, ми можемо створювати емулятори на основі доступного рівня API, ЦП та оперативної пам’яті.

Розглянемо ж більш зручний для використання емулятор Genymotion (див. рис. 5).

Для встановлення Genymotion Android Emulator на комп’ютер з операційною системою Windows необхідно виконати наступні кроки:

- встановити Android-емулятор Genymotion, завантаживши інсталяційний пакет з сайту https://www.genymotion.com/download/.

- Запустити завантажений інсталяційний файл та виконати встановлення емулятору на комп’ютер.
- Запустити емулятор після завершення установки.
- Натиснути на кнопку “Create Account” та відповідно створити собі аккаунт користувача емулятору.
- Після створення аккаунту, потрібно авторизуватись у емулятор.

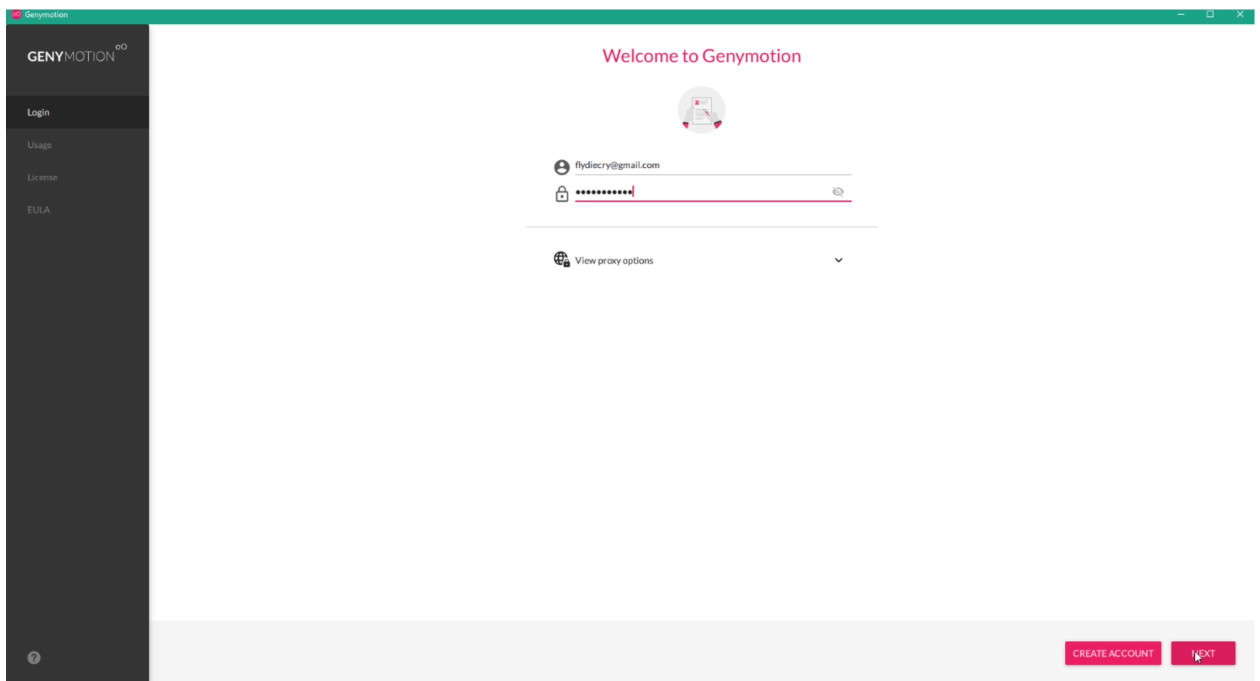


Рис. *5 Вікно авторизації Genymotion*

- Натиснути на кнопку “Personal Use”, а потім “Next” на вікні вибору ліцензії.
- Прийняти ліцензійне погодження і потрапити на головний екран емулятору.

- Щоб додати новий пристрій для емуляції потрібно натиснути на велику «+» кнопку у верхньому правому кутку (див. рис. 6).

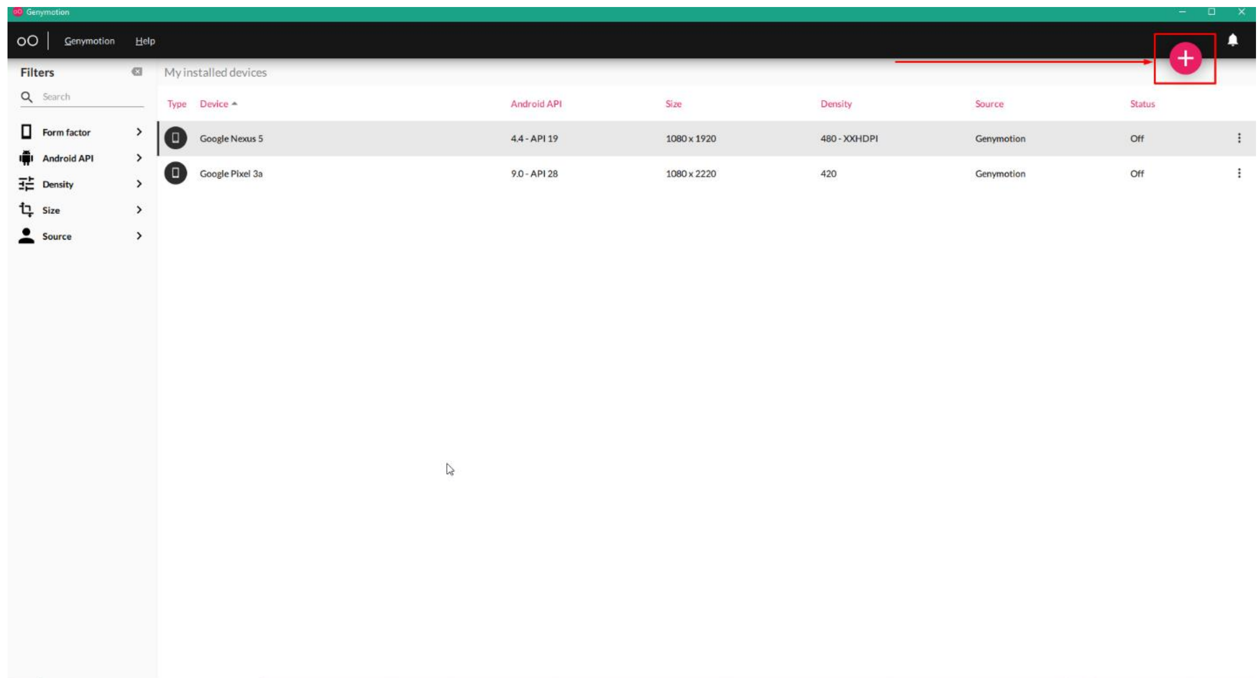


Рис. 6 *Головний екран Genymotion*

- Обрати потрібний девайс, який необхідно емулювати
- Провести налаштування віртуального девайсу під необхідні потреби, або якщо це не потрібно, клікнути на кнопку «Install», після чого новий пристрій додасться до списку ваших девайсів.
- Все, програма готова до роботи. Щоб запустити необхідний девайс для емуляції, потрібно клікнути на необхідний девайс двічі. Після чого емулятор розпочне свою роботу (див. рис. 7)

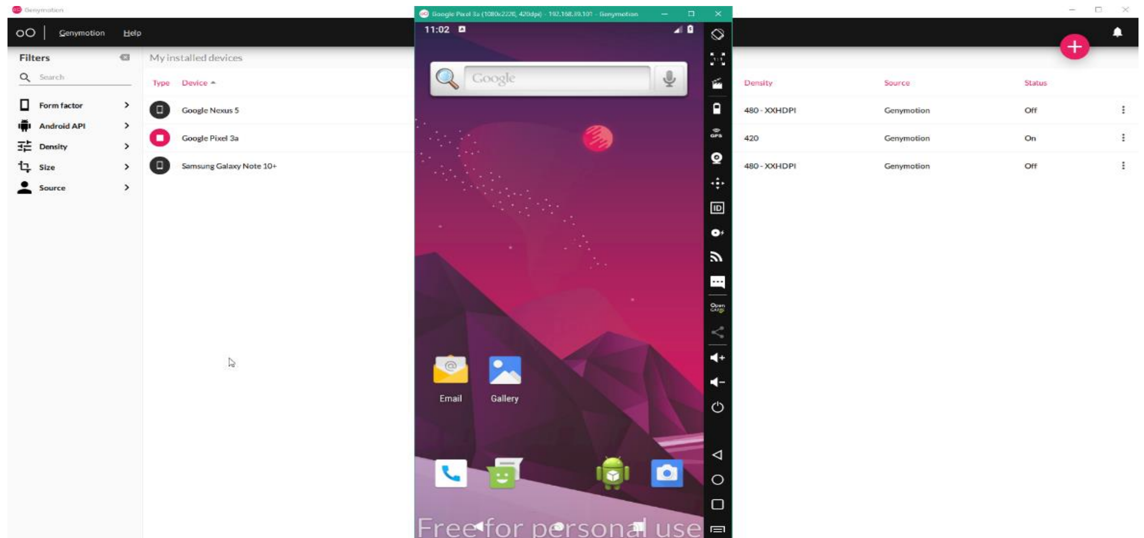


Рис. 7 *Працюючий Genymotion емулятор*

2.1.4 Підключення реальних мобільних пристроїв

Підключення реальних мобільних пристроїв до комп’ютера, з якого запускаються тести, виконується за допомогою утиліти командного рядка Android Debug Bridge (ADB), яка входить до складу Android SDK і дозволяє копіювати файли на пристрій і назад, встановлювати і видаляти програми, виконувати резервне копіювання і відновлення тощо [27].

Перш за все, на реальному пристрої потрібно включити «Режим розробника». Для цього потрібно перейти до налаштувань вашого смартфона, перейти до системної інформації про телефон та декілька разів тапнути по номеру зборки вашого смартфона, після чого з'явиться інформаційне повідомлення, що ви стали розробником.

Далі, на реальному пристрої необхідно включити налагодження по USB.

Після включення налагодження по USB, розблокуйте свій смартфон або планшет. Не використовуйте порт USB 3.0, тільки USB 2.0 при підключенні до комп'ютера. Коли пристрій підключили до комп'ютера вперше, з'явиться запит на довіру комп'ютера: поставте галочку і натисніть кнопку OK. Налагодження по USB увімкнене.

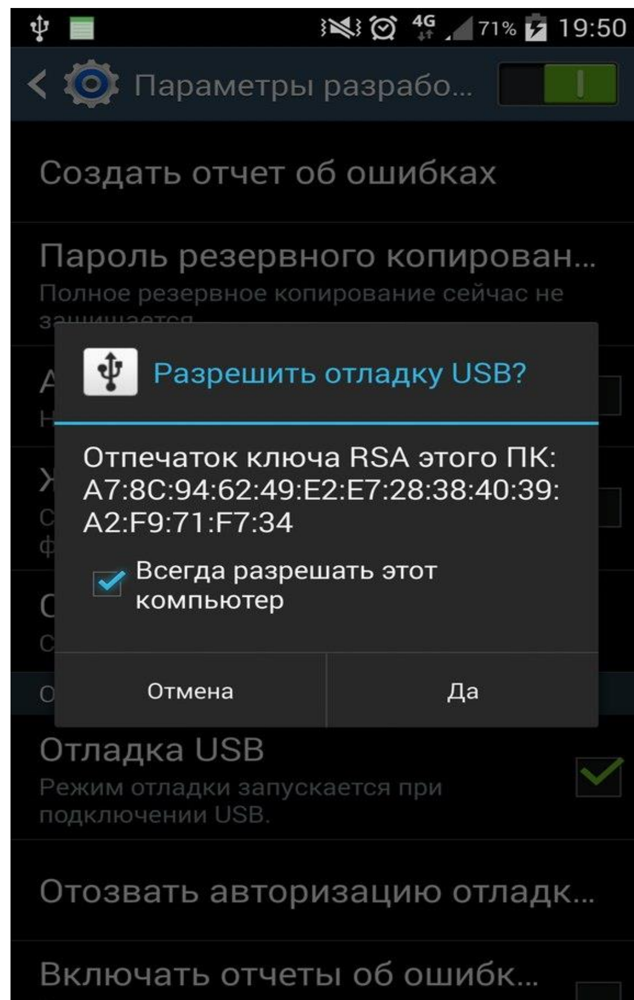


Рис. *8 Дозвіл на відлагодження*

Для того щоб перевірити чи бачить ADB ваш телефон, потрібно в консолі операціної системи комп’ютера ввести команду:

```
adb devices
```

```
E:\dev>adb devices
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
List of devices attached
02a81556e0771067        device
```

Рис. *9 adb devices для відображення девайсів у CMD*

Доречі, ім’я пристрою, отримане за допомогою adb, вказується у Desired Capabilities як `deviceName` [27, 28, 29].

Після встановлення та налаштування софту, необхідного для тестування Android, можна приступати до написання самих тестів і запуску їх на мобільному пристрої або емуляторі [30].

2.1.5 Пошук локаторів елементів інтерфейсу у Appium Inspector

Appium Inspector — це інспектор з графічним інтерфейсов для мобільних застосунків та багато іншого, що працює на окремо встановленому сервері Appium. Appium Inspector — це, в основному, просто клієнт Appium (як WebDriverIO, Appium's Java Client, Appium's Python Client і так далі) з користувацьким інтерфейсом. Використовується цей інтерфейс для того, щоб визначити, який сервер Appium використовувати, які залежності встановити, а потім взаємодіяти з елементами та іншими командами Appium після того, як ви розпочали сессію.

Основними можливостями Appium Inspector є:

- Легко визначте деталі підключення до сервера Appium та налаштуйте можливості.
- Збережіть відомості про сервер і набори можливостей для майбутніх сеансів.
- Підключайтеся до різних хмарних платформ Appium.
- Приєднуйся до наявного сеансу Appium через його ідентифікатор (ID).
- Перевірте знімок екрана та source мобільного додатка. (Цей інспектор призначений для роботи з iOS та Android. Інші платформи Appium також можуть працювати, але вони, ймовірно, не працюватимуть без деяких оновлень коду).
- Виберіть елементи, натиснувши на них на знімку екрана.
- Взаємодія з елементами (click, sendKeys, clear).
- Отримайте список запропонованих стратегій та селекторів локатора елементів, які будуть використовуватися у ваших сценаріях.
- Порівняйте швидкість різних стратегій пошуку елементів.

- Запускати та зупиняти режим «запису», який перетворює ваші дії в інспекторі на зразки коду, які можна використовувати у своїх сценаріях.
- Натисніть на екран у довільному місці.
- Виконайте жест пальцем.
- Перемикайтеся в режими веб-контексту та взаємодійте з веб-елементами.
- Перевірте власні стратегії локатора.
- Отримайте доступ до величезної бібліотеки дій Appium для виконання простим клацанням миші, включаючи надання власних параметрів.

Як виглядає графічний інтерфейс Appium Inspector можна побачити на рисунку 10.

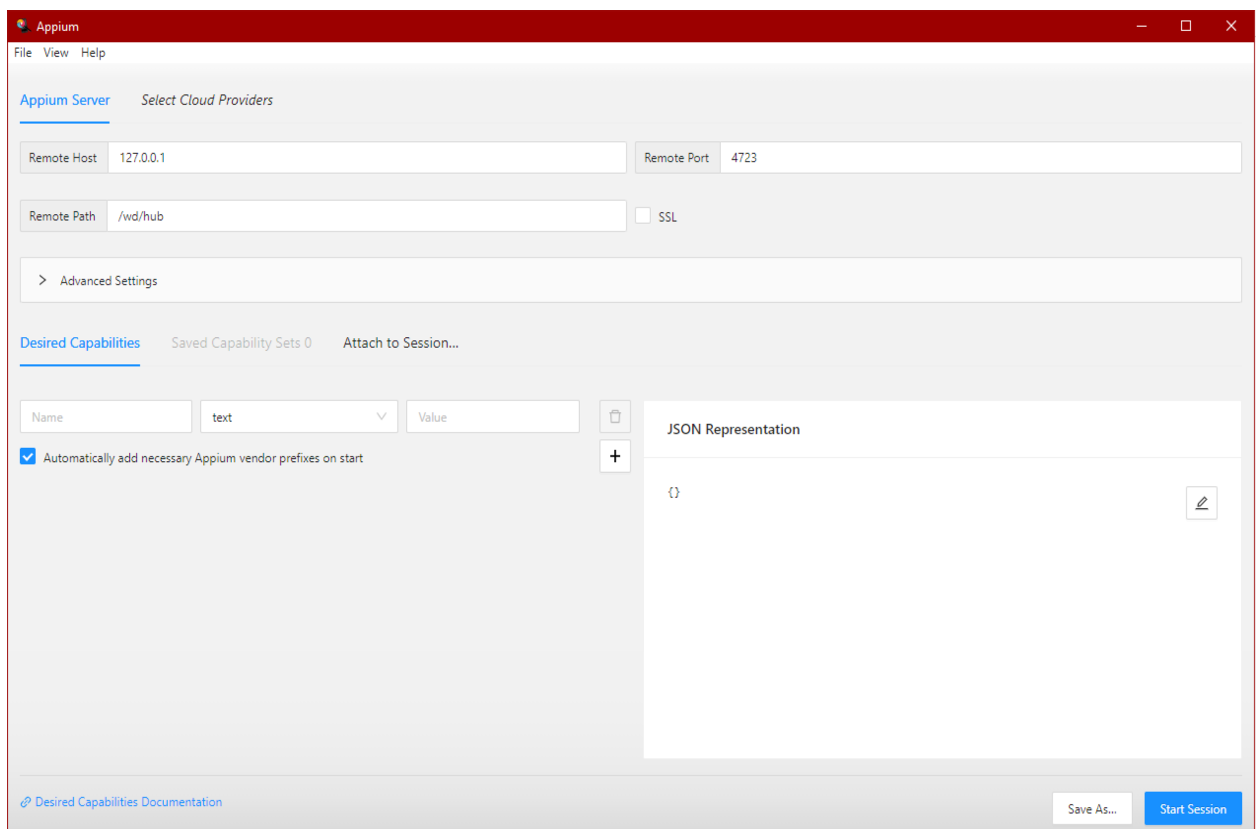


Рис. *10 Appium Inspector GUI*

Для того, щоб налаштувати Appium Inspector на певну сессію, нам необхідно підключити фізичний девайс до персонального компьютера, після чого

запустити Appium Server та налаштувати його на локальну сессію, а саме вказати у полі Host локальну айпі мережу (127.0.0.1) (див. рис. 11).

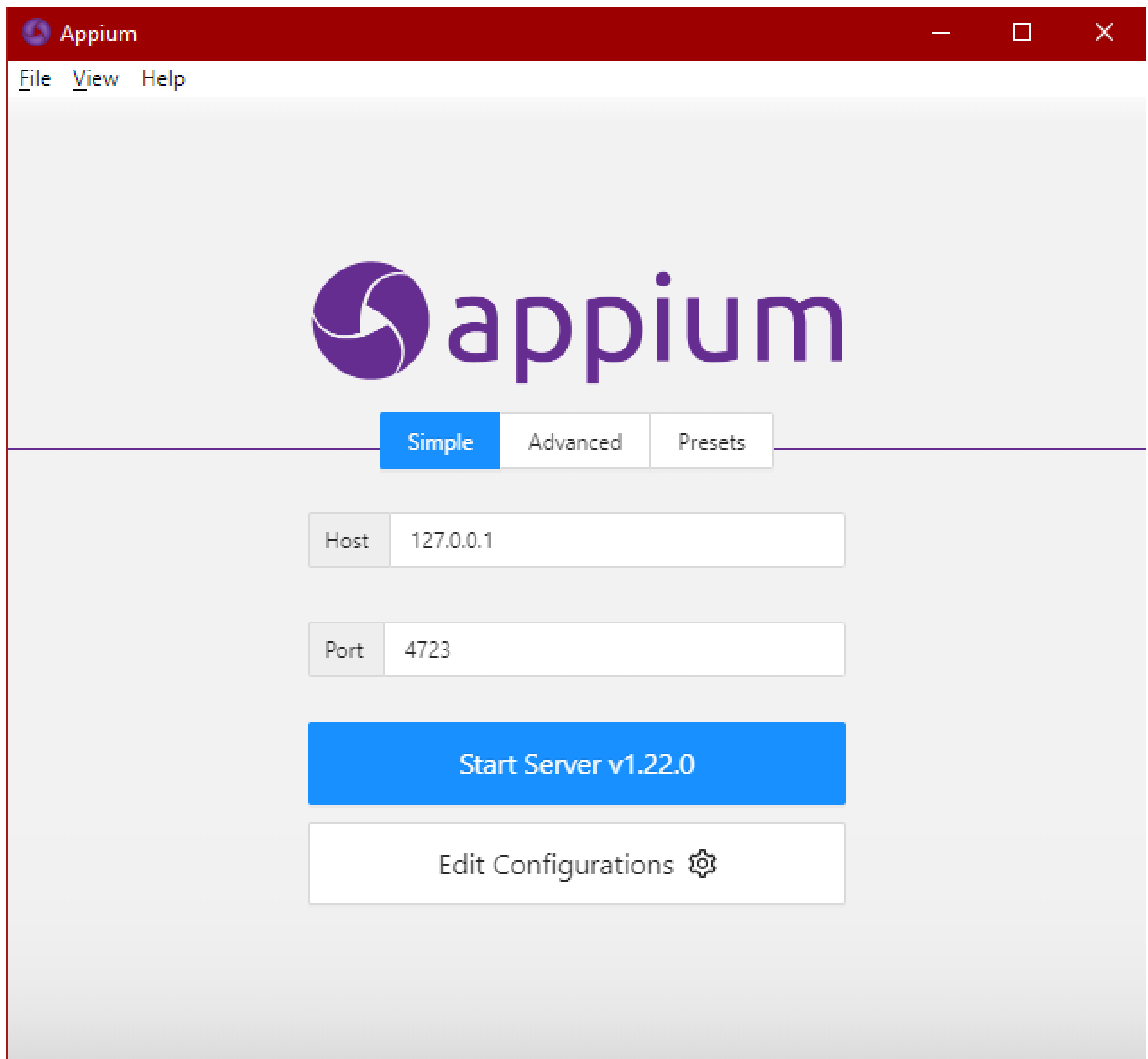


Рис. *11 Налаштування Appium Server*

Після чого потрібно натиснути на кнопку «Start Server», та перейти до вікна Appium Inspector. Про всяк випадок можна перевірити, чи відображається підключений девайс у системі прописавши команду adb devices у CMD.

У вікні Appium Inspector перш за все нам необхідно налаштувати Desired Capabilities. У цьому вікні нам необхідно описати залежності, в залежності від девайсу, який ми будемо використовувати (див. рис. 12).

В залежності від необхідної нам сесії, desired capabilities будуть трохи відрізнятись. Наприклад для того, щоб шукати необхідні локатори у браузері, у залежностях потрібно вказати browserName, а якщо нам необхідно шукати локатори у нативному чи гібридному застосунку, потрібно вказати appPackage та appActivity.

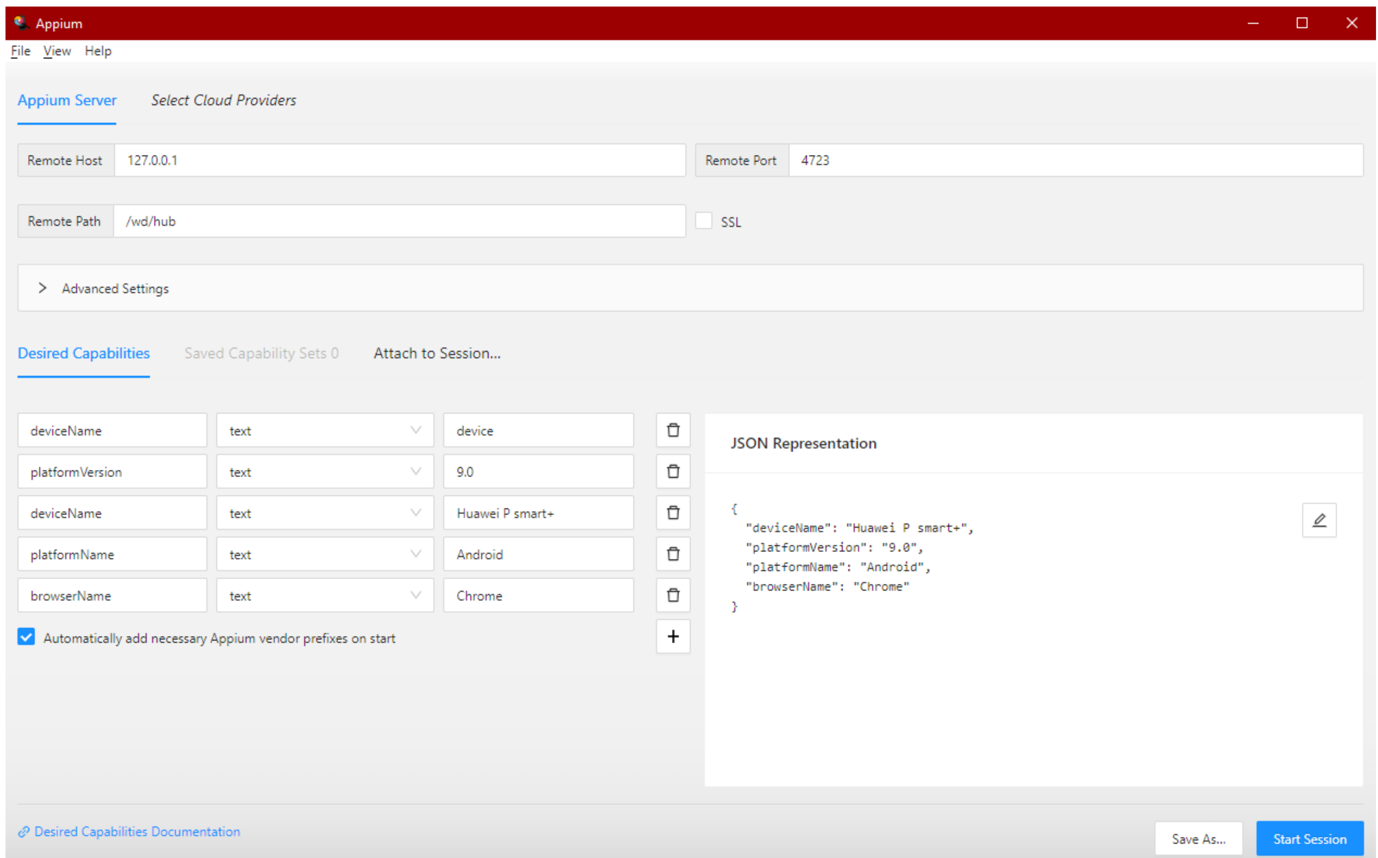


Рис. *12 Налаштування desired capabilities для веб-застосунку у Appium Inspector*

Після того, як залежності налаштовані, потрібно натиснути на кнопку «Start Session», трошки зачекати поки запуститься сессія і після чого ми побачимо екран Appium Inspector у якому вже відображається снімок з екрана нашого підключенного девайсу (див. рис. 13).

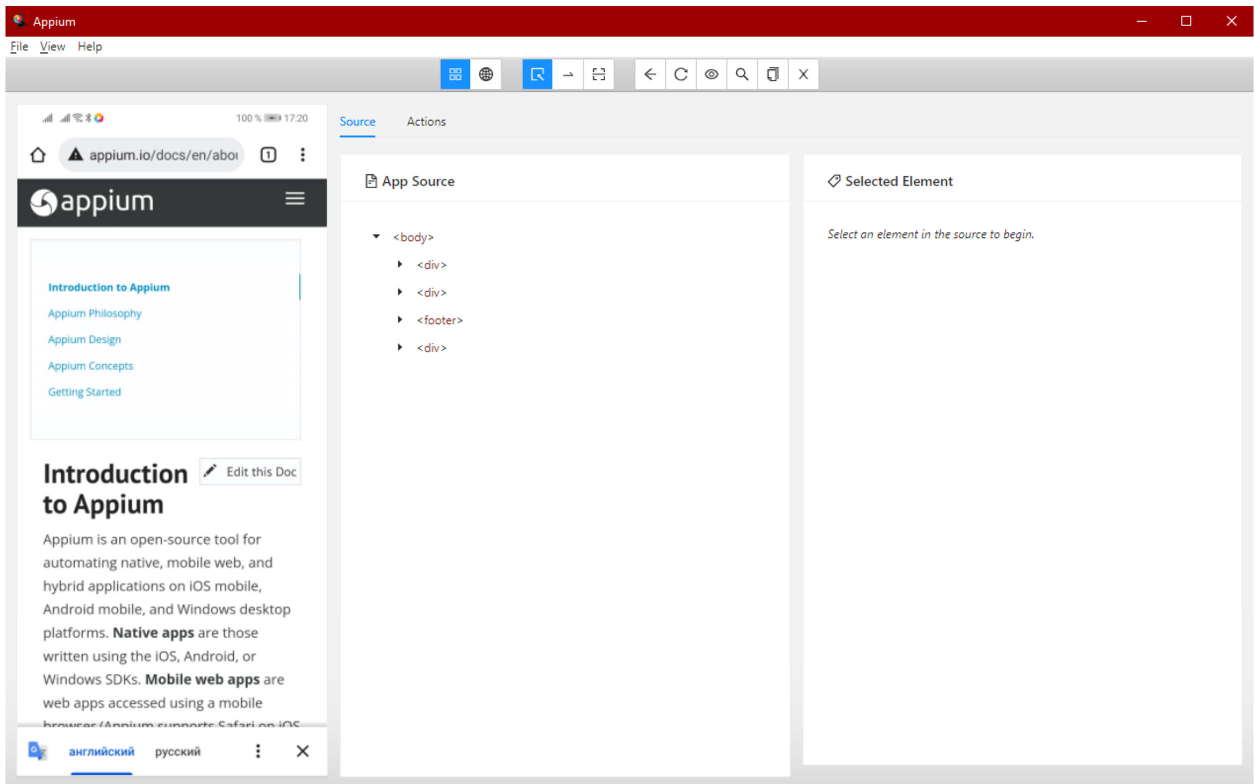


Рис. *13 Відкрита сесія Appium Inspector*

Після того, як Appium Inspector був налаштований, можно приступати до роботи. Appium Inspector дозволяє знаходити локатори як у нативних, так і в гібридних та веб-застосунках. Для того, щоб вибрати необхідну конфігурацію, потрібну клікнути кнопку App Mode у контекстному меню інспетора зверху (див. рис. 14).

Один із найсуттєвіших недостатків Appium Inspector є те, що дані з екрана не оновлюються у реальному часі, а це означає, що при переході на нову сторінку, чи новий екран, або навіть при скроллі мобільної сторінки чи екрану нам потрібно постійно оновлювати екран для того, щоб знайти необхідні локатори.

Для того, щоб оновити дані з екрана мобільного пристрою, необхідно клікнути на кнопку «Refresh Source & Screenshot», що також знаходиться зверху у контекстному меню інспектора (див. рис. 15).

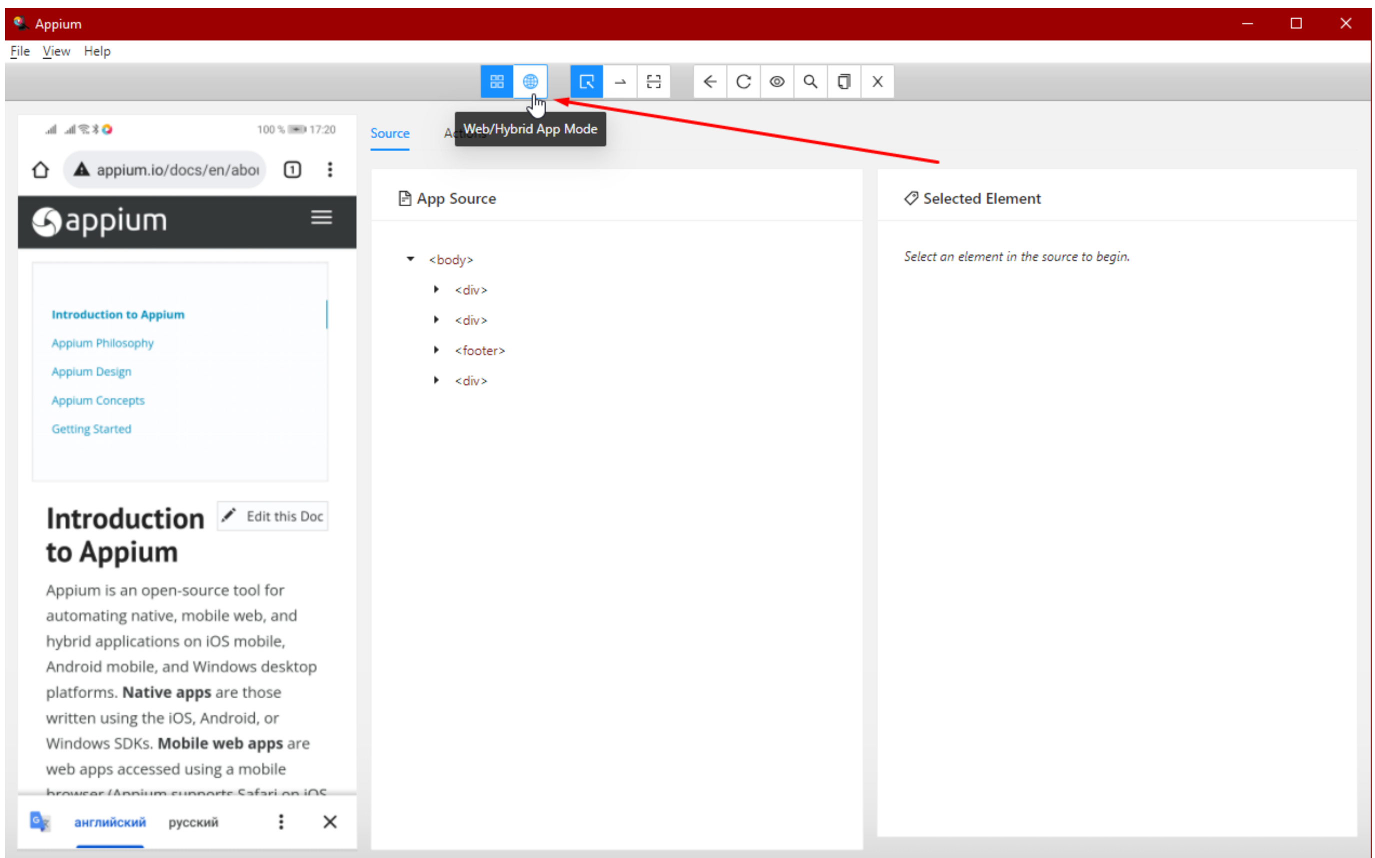


Рис. *14 Конфігурація Appium Inspector під необхідний застосунок*

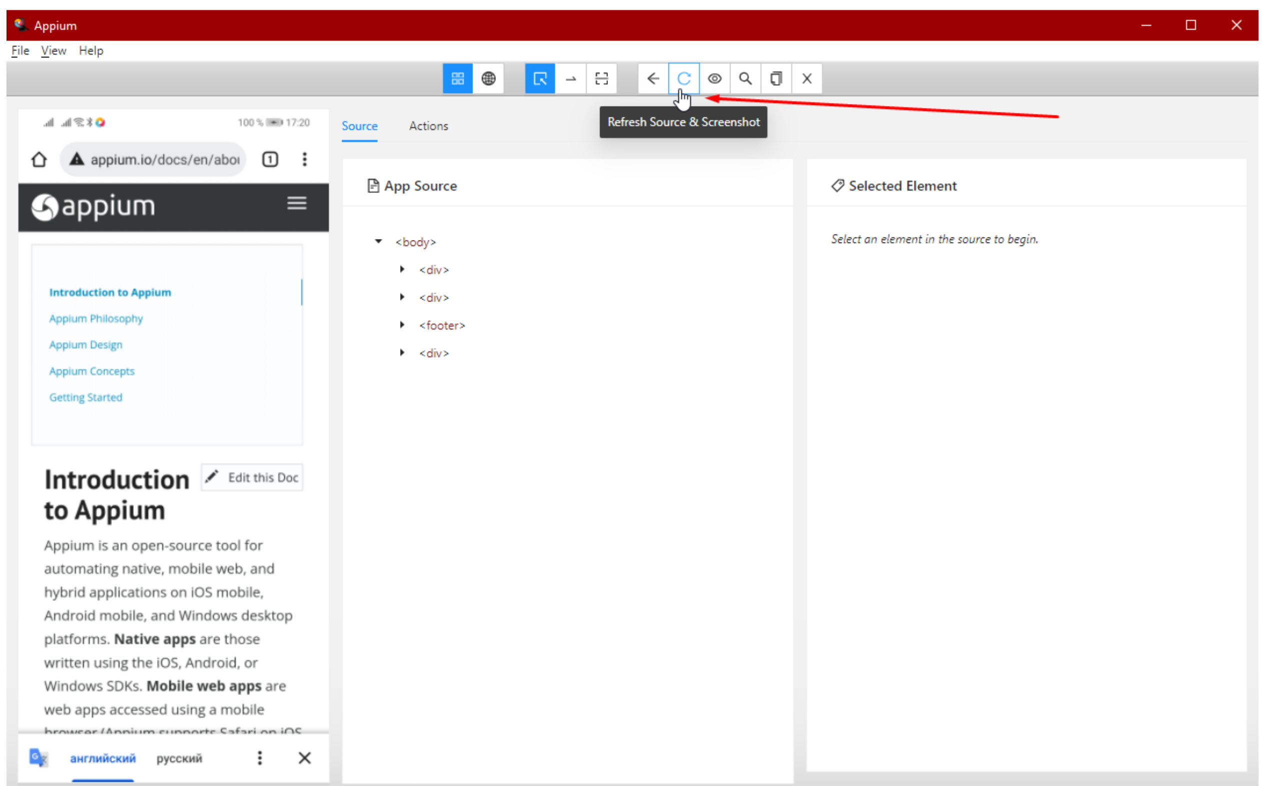


Рис. *15 Розміщення кнопки «Refresh Source & Screenshot»*

Appium Inspector дозволяє дивитись інформацію елементів за допомогою можливості функції «Select Elements», а також підтримує можливість використовувати тап та свайпи у мобільному пристрої. У більшості випадків якщо ви перевіряєте веб-застосунок, ви, певне, не будете використовувати Appium Inspector для пошуку необхідних вам локаторів по тій причині, що час, витрачений на оновлення даних з екрану буде дуже великим, аніж якщо просто відкрити необхідну сторінку в браузері та шукати локатори безпосередньо через Dev Tools. Плюс до всього, локатори, які Appium Inspector дозволяє шукати через веб-застосунки іноді відображаються з помилками, що також негативно вплине на час, витрачений на роботу.

Також для того, щоб перевіряти функціонал будь-якого застосунку, потрібно розробити тестові сценарії. Мною були розроблені тестові сценарії для нативного, веб та гібридного застосунку, які я створив та реалізував у середовищі розробки.

Під час розробки тестових сценаріїв у застосунках моєю основною метою була розробка поизитивних тестових сценаріїв тобто коли користувач використовує валідні дані під час виконання певної дії, та негативних тестових сценаріїв, тобто коли користувач використовує навпаки, невалідні дані під час виконання певної операції або дії.

Далі в таблиці 4 представлено опис безпосередньо тих тестових сценаріїв, що розроблялися для тестування застосунків інтернет-магазину Elitan:

Таблиця *4*

Тестові сценарії застосунків інтернет-магазину Elitan

ID	Передумова	Опис	Кроки	Очікуваний результат
1	Веб-сайт відкритий, користувач знаходиться на екрані авторизації	Перевірка форми авторизації з дійсними даними	1. Ввести в поле "Email" дійсні дані	Поле акцентовано, дані відображаються
			2. Ввести в поле "Пароль" дійсні дані	Поле акцентовано, дані відображаються зашифрованими у вигляді черних точок
			3. Натисніть кнопку "Войти"	Кнопка красного кольору. Происходит переход на сторінку "Scholarship Feed", авторизация пройшла успішно
2	Веб-сайт відкритий, користувач знаходиться на екрані авторизації	Перевірка форми авторизації з використанням невалидної пошти	1. Ввести в поле "Email" невалідні дані (Без "@", без домену, тд)	Поле акцентовано, дані відображаються
			2. Ввести в поле "Пароль" дійсні дані	Поле акцентовано, дані відображаються зашифрованими у вигляді черних точок
			3. Натисніть кнопку "Войти"	Повідомлення про введення невалідних даних відображається на екрані
3	Веб-сайт відкритий, користувач знаходиться на екрані реєстрації	Перевірка реєстрації на сайті з використанням валідних даних	1. Заповнити всі обов'язкові поля валідними даними	Поля акцентовані, дані відображаються. Дані у полі "Пароль" відображаються зашифрованими у вигляді чорних крапок.
			2. Клікнути на кнопку "Реєстрація"	Відображається клік по кнопці, відбувається редирект на сторінку "Мій обліковий запис", інформаційне повідомлення "Ласкаво просимо" відображається на екрані

Продовження таблиці *4*

4	Веб-сайт відкритий, користувач знаходиться на екрані реєстрації	Перевірка реєстрації на сайті з використанням невалідних даних	1. Заповнити всі обов'язкові поля невалідними даними (використовуючи спец. символи, цифри, порушення граничних значень тощо)	Поля акцентовані, дані відображаються. Дані у полі "Пароль" відображаються зашифрованими у вигляді чорних крапок. Текстові поля не підтримують символи та цифрові значення, текстові поля обмежені для введення безлічі символів.
			2. Клікнути на кнопку "Реєстрація"	Спрацьовує обробка винятків із повідомленням про введеність невалідних даних, усі поля з помилками підсвічені червоним кольором
5	Веб-сайт відкритий, користувач зареєстрований та знаходиться на сторінці "Профіль"	Зміна контактних даних користувача	1. Тапнути по полю "Ім'я"	Поле акцентовано, ім'я задане раніше відображається
			2. Видалити дані та заповнити поле валідним ім'ям	Поле акцентовано, дані, що вводяться, відображаються
			3. Тапнути по полю "Прізвище"	Поле акцентовано, прізвище, задане раніше, відображається
			4. Видалити дані та заповнити поле валідним прізвищем	Поле акцентовано, дані, що вводяться, відображаються
			5. Тапнути по полю "Ім'я, що відображається"	Поле акцентовано, ім'я, задане раніше, відображається
			6. Видалити дані та заповнити поле валідним нікнеймом	Поле акцентовано, дані, що вводяться, відображаються
			7. Клікнути на кнопку "Зберегти зміни"	Повідомлення про те, що введені дані збереглися, відображається
6	Веб сайт відкритий, користувач зареєстрований та авторизований. Користувач знаходиться на сторінці "Головна".	Перевірка додавання товару до кошика	1. Відкрити будь-який товар, що знаходиться на головній сторінці	Відкритий товар відображається у тому ж вікні, інформація про товар відображається

РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОДУКТУ

В даному розділі описується безпосередньо програмна реалізація продукту з лістингами, описом реалізації тестових сценаріїв та інше.

3.1 Програмна реалізація тестування веб-застосунку

Перш за все у реалізації тестів для веб-застосунку необхідно сторити клас, наприклад `ElitanWebTest`, та в ньому створити змінну AndroidDriver командою `protected static AndroidDriver<WebElement> driver;`

Після чого необхідно вказати, який девайс ми будемо використовувати, а саме фізичний, чи з емулятора [15, 16]. Це можна описати командою `private static final boolean isRealDevice = true;` де true — це підтвердження реального девайсу, а false — це девайс, який емулюється.

Коли змінна AndroidDriver була створена, потрібно визначити метод `@BeforeAll`.

`@BeforeAll` використовується для сигнала про те, що анотований метод повинен бути виконаний перед усіма тестами в поточному тестовому класі.

У цьому методі ми переходимо до налаштувань залежностей, тобто DesiredCapabilities, про які мовилось раніше [17, 18].

У залежностях ми повинні прописати умову для визначення залежностей девайсу фізичного та емульованого:

Лістинг 2 *Умова визначення залежностей фізичного та емульованого девайсу*

```
@BeforeAll
public static void setUpClass() throws MalformedURLException {
```

```
     DesiredCapabilities capabilities = new
DesiredCapabilities();
             if (isRealDevice) {
     capabilities.setCapability(MobileCapabilityType.DEVICE_
NAME, "device");

     capabilities.setCapability(MobileCapabilityType.PLATFOR
M_VERSION, "9.0");
             } else {

capabilities.setCapability(MobileCapabilityType.DEVICE_NAME
, "Android10Phone");

capabilities.setCapability(MobileCapabilityType.PLATFORM_VE
RSION, "10.0");
             }

capabilities.setCapability(MobileCapabilityType.PLATFORM_NA
ME, "Android");

capabilities.setCapability(MobileCapabilityType.AUTOMATION_
NAME, "UiAutomator2");

capabilities.setCapability(MobileCapabilityType.BROWSER_NAM
E, "Chrome");
```

На лістингу можна побачити, що для веб-застосунку використовуються такі залежності як: deviceName, platformName, platformVersion, browserName. Відповідно:

- deviceName — назва мобільного пристрою
- platformName — назва платформи, на якій буде проводитись тестування

- platformVersion — версія операційної системи
- browserName — назва веб-браузеру, на якому буде виконуватись тестування

Після чого потрібно виконати ініціалізацію AndroidDriver під нашу локальну машину:

Лістинг *3 Ініціалізація AndroidDriver*

```
driver = new AndroidDriver<>(new URL("http://127.0.0.1:4723/wd/hub"), capabilities);
driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
```

На лістингу ми можемо побачити те, що ми вказуємо IP мережу при ініціалізації AndroidDriver, потрібно це для того, щоб вказати, що всі тести будуть виконані на, безпосередньо, локальній машині, тобто на вашому комп'ютері [19, 20].

Після того як AndroidDriver був ініціалізований та необхідні залежності прописані, треба перейти до реалізації тестових сценаріїв у вигляді програмного коду. Для цього нам потрібно об'явити команду `@Test`.

`@Test` використовується для сигнала про те, що анотований метод є методом тестування. Важилим фактом є те, що написати навіть декілька тестів під сигналом `@Test` — неможливо, тому треба кожний тест сигналізувати як тестовий метод, відповідно кожний метод буде відмічений командою `@Test`.

Основною задачею першого тестового сценарію була авторизація на сайті з використанням валідних даних.

Для цього у коді потрібно оголосити три строкових змінні, в які ми передамо валідні дані користувача:

Лістинг 4 *Створення строкових змінних з валідними даними користувача*

```
@Test
    void testLogin() {
        /*Registered user login and password*/
        String email = "usertest@gmail.com";
        String password = "testtest";
        String username = email.substring(0, email.indexOf('@'));
```

Після чого за допомогою WebDriver нам необхідно перейти до необхідного для тестування веб-сайту та за допомогою локаторів виконати авторизацію.

Лістинг 5 *Реалізація авторизації на сайті за допомогою локаторів*

```
driver.get("https://elitan.com.ua/");
        driver.findElement(By.linkText("Вход/ Регистрация")).click();
        /*Opens authorization page /my-account*/

driver.findElement(By.cssSelector("#username")).sendKeys(email);

driver.findElement(By.cssSelector("#password")).sendKeys(password);

driver.findElement(By.cssSelector(".woocommerce-form-login__submit")).click();
```

На приведеному лістингу можна побачити, що перш за все ми передаємо команду драйверу про те, що нам необхідно перейти на наш сайт для тесту-

вання, після чого ми шукаємо гіперпосилання на сторінку авторизації за допомогою команди `driver.findElement(By.linkText)` та клікаємо по ній за допомогою команди `.click();`

Потім ми за допомогою пошуку локаторів визначаємо поля для вводу Email та паролю, вводимо в них валідні дані за допомогою команди `.sendKeys`, де в параметри передаємо строкові змінні, що зберігають у собі необхідні для авторизації дані та шукаємо кнопку для входу в облікову сторінку користувача. Після чого клікаємо по кнопці за допомогою команди `.click()` та потрапляємо до екрану «Мій профіль», де можемо побачити інформацію про авторизованого користувача.

Для того, щоб верифікувати фактичні результати тесту з очікуваними, необхідно використати команду `String expected`, яку можна побачити у лістингу нижче, після чого за допомогою локаторів знайти щось, що буде вказувати на вдале завершення тесту та наш очікуваний результат, та за допомогою ассертів отримати повідомлення про те, що очікуваний результат співпадає за фактичним [21, 22].

Лістинг 6 *Реалізація порівняння фактичного результату з очікуваним*

```
String expected = "Добро пожаловать, " + username + " (не " + username + "? Выйти)";
        WebElement messageElement = driver.findElement(
                By.cssSelector(".woocommerce-MyAccount-content > p:nth-child(2)"));
        Assertions.assertEquals(expected, messageElement.getText());
```

Коли фактичний результат було порівнювано з очікуваним і результати співпадають, ми можемо створювати нові тести за тим же способом, як і перший описаний тест. Для цього потрібно лише зробити анотацію `@Test`, а потом описувати метод, який буде виконувати тестування.

Далі я опишу метод, який виконає тестування авторизації на сайті з використанням невалідних даних.

Лістинг 7 *Реалізація тесту авторизації на сайті з використанням невалідних даних*

```
@Test
void testInvalidLogin() {
    /*Registered user login and password*/
    String email = "thereissome@textgmail.com";
    String password = "qwerty";
    String username = email.substring(0,
email.indexOf('@'));

    driver.get("https://elitan.com.ua/my-
account/");

driver.findElement(By.xpath("//input[@id='username']")).sen
dKeys(email);

driver.findElement(By.xpath("//input[@id='password']")).sen
dKeys(password);

driver.findElement(By.cssSelector("#rememberme")).click();

driver.findElement(By.cssSelector(".woocommerce-form-
login__submit")).click();

    String expected = "ERROR";
    WebElement messageElement = driver.findElement(
            By.xpath("//*[@id=\"post-
7\"]/div/div/div/div/div[1]/ul/li/strong"));
```

```
            Assertions.assertEquals(expected,
messageElement.getText());
        }
```

На лістингу потрібно звернути увагу перш за все на те, що буде зберігатися у строкових даних, а саме в них будуть знаходитися невалідні дані, при вводі яких ми будемо очікувати виключення про введення невалідних даних.

Код у даній реалізації не буде дуже відрізнятись від тесту логіну з валідними даними. Другою відмінністю стане лише те, що у порівнянні фактичного результату з очікуваним ми будемо шукати виключення з повідомленням про введення невалідних даних у поля.

Далі виконаємо реалізацію тестового сценарію для реєстрації на сайті з використанням валідних даних [23,24].

Лістинг *8 Реалізація тесту реєстрації на сайті з використанням валідних даних*

```
        @Test
        void testValidRegistration() {
            String name = "TestQA";
            String surname = "QATest";
            String phone = "0665711464";
            String email = "flydiecry+211103@gmail.com";
            String password = "Test1234!";
            //String username = email.substring(0, email.in-
dexOf('@'));

            driver.get("https://elitan.com.ua/my-ac-
count/");
            driver.findEle-
ment(By.xpath("//*[@id=\"reg_billing_first_name\"]")).send-
Keys(name);
```

```
            driver.findEle-
ment(By.xpath("//*[@id=\"reg_billing_last_name\"]")).send-
Keys(surname);
            driver.findEle-
ment(By.xpath("//*[@id=\"reg_billing_phone\"]")).send-
Keys(phone);
            driver.findEle-
ment(By.xpath("//*[@id=\"reg_email\"]")).sendKeys(email);
            driver.findEle-
ment(By.xpath("//*[@id=\"reg_password\"]")).sendKeys(pass-
word);
            driver.findElement(By.xpath("//*[@id=\"cus-
tomer_login\"]/div[2]/form/p[6]/button")).click();

    //        /*Для успішної реєстрації*/
    //        String expected = "Выйти";
    //       WebElement messageElement = driver.findElement(
    //                By.cssSelector("#post-7 > div > div >
div > div > div > div > p:nth-child(2) > a"));
    //        String messageText = messageElement.getText();

            /*Для реєстрації вже зареєстрованого користу-
вача*/
            String expected = "Ошибка:";
            WebElement messageElement = driver.findElement(
                    By.xpath("//*[@id=\"post-
7\"]/div/div/div/div/div/div[1]/ul/li"));
            String[]    messageWords    =    messageEl-
ement.getText().split( "\\s");
            String messageText = messageWords[0];

            Assertions.assertEquals(expected, messageText);
        }
```

На даному лістингу ми можемо побачити те, що для реєстрації нам необхідно розширити список даних, необхідних для заповнення полів, а також розширується пошук локаторів та дій, які необхідно виконати для проходження реєстрації на сайті [25].

Для початку нам необхідно передати дані у строкові змінні, які ми будемо вводити до полів реєстрації. На сайті представлені обов'язкові поля ім'я, прізвища, телефону, поштової адреси та паролю (див. рис. 16).

Після того, як ми створили строкові змінні та передали в них інформацію про те, які дані необхідно буде ввести, ми розпочинаємо поетапну реалізацію кроків для проходження реєстрації на сайті.

Для цього ми знаходимо по локаторам всі обов'язкові поля та посилаємо в них дані, які ми зберігали раніше. Після чого, клікаємо по кнопці «Регистрация» та переходимо до етапу порівняння результатів.

Нашим основним очікуваним результатом буде те, що на сторінці з'являється кнопка «Выйти» (див. рис. 17), тому перевіряємо наявність цієї кнопки на сайті і якщо кнопка відображається, то тест вважається повністю завершеним.

Регистрация

(для новых клиентов)

Имя*

Фамилия*

Телефон*

Email *

Пароль *

Ваши личные данные будут использоваться для упрощения вашей работы с сайтом, управления доступом к вашей учётной записи и для других целей, описанных в нашей политика конфиденциальности.

Регистрация

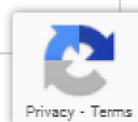


Рис. *16 Відображення обов'язкових полів реєстрації*

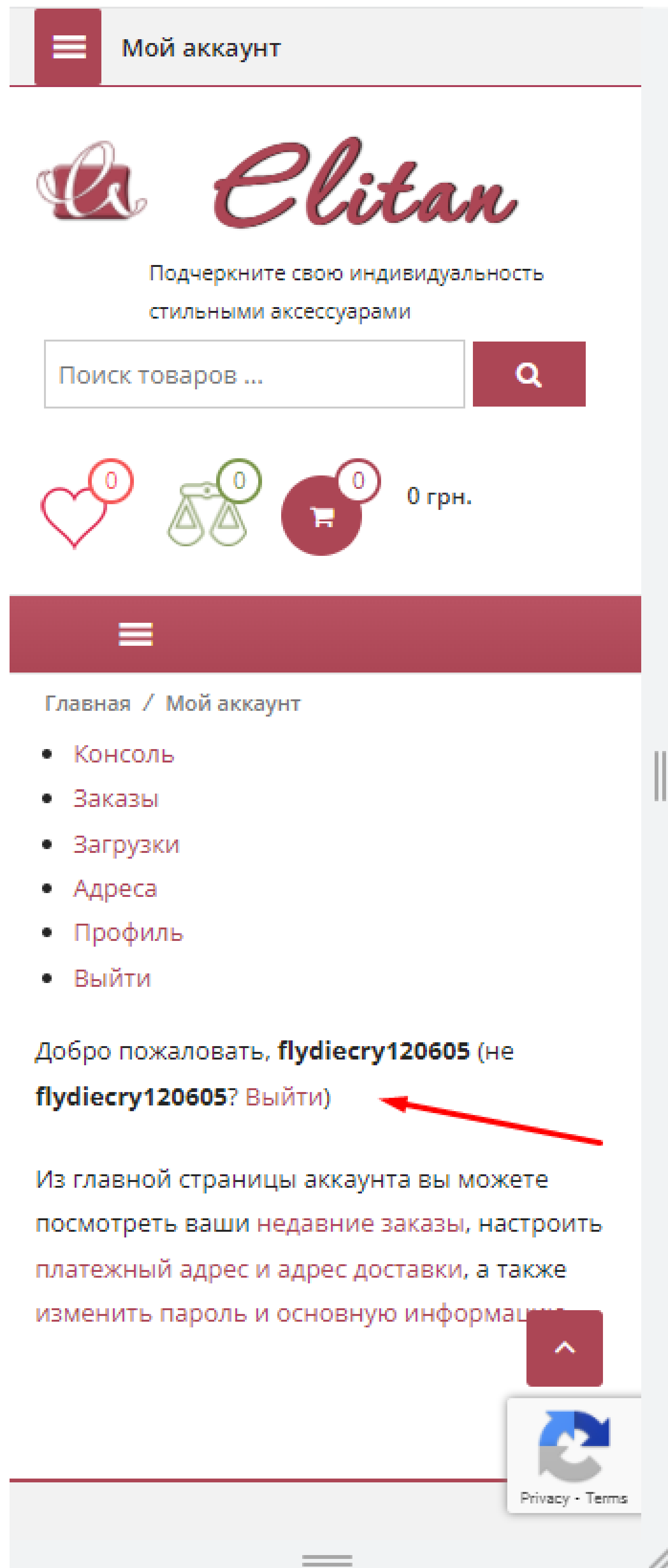


Рис. *17 Кнопка «Вийти» є очікуваним результатом*

Обов'язковим етапом є перевірка реєстрації з пропущеними полями. Необхідно це для того, щоб перевірити, чи передбачає веб-застосунок виключення на випадок не заповнених полів.

Реалізація такого тестового сценарію буде виглядати таким чином:

Лістинг 9 *Реалізація тесту реєстрації з пропущеними обов'язковими полями*

```
    /**
     * Tests registration with Empty Fields
     */
    @Test
    void testInvalidRegistrationEmptyFields() {
        driver.get("https://elitan.com.ua/my-
account/");

driver.findElement(By.xpath("//*[@id=\"customer_login\"]/di
v[2]/form/p[6]/button")).click();

        String expected = "Ошибка:";
        WebElement messageElement = driver.findElement(
                By.xpath("//*[@id=\"post-
7\"]/div/div/div/div/div[1]/ul/li"));
        String[]            messageWords             =
messageElement.getText().split( "\\s");
        String messageText = messageWords[0];

        Assertions.assertEquals(expected, messageText);
    }
```

Даний тест відрізняється від тесту реєстрації тим, що замість заповнення полів реєстрації, ми просто клікаємо по кнопці «Регистрация» та фіксуємо виключення про незаповненість полів. Очікуваним результатом у цьому тесті

стане ключове слово «Ошибка» (див. рис. 18), яке і повідомить нам про те, що веб-сайт готовий до таких негативних сценаріїв користувача.

Главная / Мой аккаунт

Ошибка: Пожалуйста, укажите действующий адрес электронной почты.

Авторизация

(для зарегистрированных клиентов)

Имя пользователя или email *

Пароль *

Войти

Запомнить меня

Забыли свой пароль?

Регистрация

(для новых клиентов)

Имя*

Privacy - Terms

Рис. *18 Повідомлення про помилку є очікуваним результатом теста*

Наступним етапом необхідно виконати перевірку можливості редагування інформації про користувача.

Реалізація виглядає наступним чином:

Лістинг *10 Реалізація тесту перевірки функціональної можливості зміни котнктної інформації користувача*

```
    /**
     * Tests changing contact info
     */
    @Test
    void changeContactInfo() {
        String email = "usertest@gmail.com";
        String password = "testtest";
        String    username    =    email.substring(0,
email.indexOf('@'));

        driver.get("https://elitan.com.ua/my-
account/");

driver.findElement(By.xpath("//input[@id='username']")).sen
dKeys(email);

driver.findElement(By.xpath("//input[@id='password']")).sen
dKeys(password);

driver.findElement(By.cssSelector("#rememberme")).click();

driver.findElement(By.cssSelector(".woocommerce-form-
login__submit")).click();
        driver.findElement(By.cssSelector(
                "#post-7 > div > div > div > div > div >
nav    >    ul    >    li.woocommerce-MyAccount-navigation-
```

```
link.woocommerce-MyAccount-navigation-link--edit-account >
a")).click();

            String name = "Tessst";
            String surname = "QATestertest";
            String dispname = "Testirovshichek";


driver.findElement(By.xpath("//*[@id=\"account_first_name\"
]")).clear();

driver.findElement(By.xpath("//*[@id=\"account_first_name\"
]")).sendKeys(name);

driver.findElement(By.xpath("//*[@id=\"account_last_name\"]
")).clear();

driver.findElement(By.xpath("//*[@id=\"account_last_name\"]
")).sendKeys(surname);

driver.findElement(By.xpath("//*[@id=\"account_display_name
\"]")).clear();

driver.findElement(By.xpath("//*[@id=\"account_display_name
\"]")).sendKeys(dispname);
            driver.findElement(By.cssSelector("#post-7 >
div > div > div > div > div > div > form > p:nth-child(9) >
button")).click();
            String expected = "Профиль успешно изменён.";
            WebElement messageElement =
driver.findElement(By.cssSelector("#post-7 > div > div > div
> div > div > div > div > div"));
```

```
            Assertions.assertEquals(expected,
messageElement.getText());
        }
```

В даному тесті ми користали нову команду, яку не використовували раніше. Як відомо, на більшості, якщо не на всіх сайтах, коли ми заходимо на сторінку профіля користувача, ми бачимо наші дані. У даному випадку, якщо ми будемо як і раніше використовувати `sendKeys();` та відправляти у поле збережені раніше у строкових змінних дані, то тест буде виглядати погано, тому що старі дані не будуть видалятися, а нові дані будуть додаватись до раніше створених.

Щоб уникнути такої ситуації, перед тим, як посилати дані у текстове поле, необхідно виконати команду `driver.findElement(*пошук_елементу*).clear();`.

Ця команда виконає повну очистку текстового поля перед тим, як виконувати з ним якісь дії.

Після чого можна посилати в поле дані з інформацією, яку ми зберігали у строкових змінних.

Як результат, ми авторизуємось на сайті, після чого переходимо до сторінки профілю користувача, у кожному полі спочатку видаляємо інформацію, а потім заповнюємо її нашими даними, та клікаємо по кнопці «Сохранить изменения». Після чого ми перевіряємо фактичний результат з очікуваним.

В нашому випадку очікуваним результатом стане інформаційне повідомлення про те, що дані профіля успішно збережені (див. рис. 19).

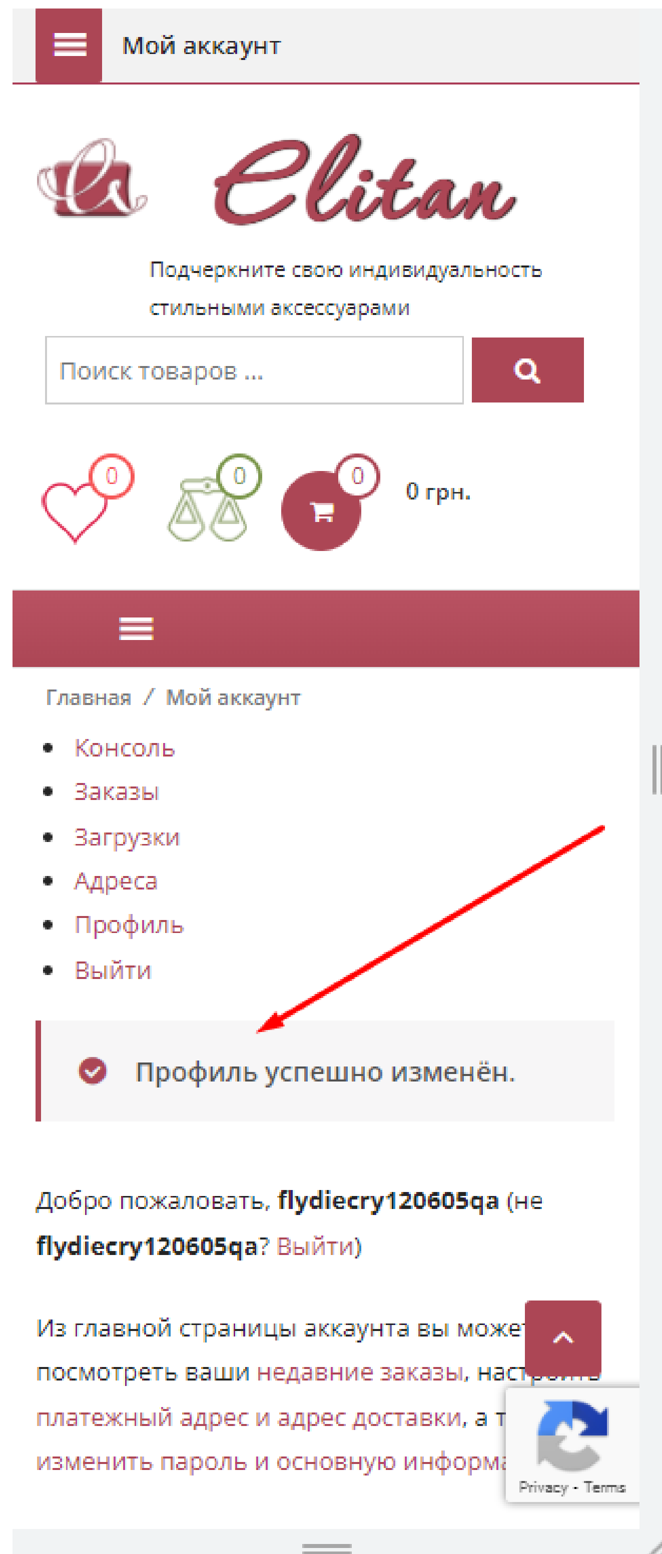


Рис. *19 Очікуваний результат тесту зміни інформації користувача*

Наступною реалізацією тестового сценарію стала перевірка можливості додавання товару до кошика.

Реалізація виглядає наступним чином:

Лістинг *11 Реалізація функціонального тесту додавання товару до кошика*

```
    /**
     * Tests adding item to the cart
     */
    @Test
    void addItemToTheCart() {
        WebElement addToCartButton =
driver.findElement(By
                .xpath("//*[@id=\"post-
24\"]/div/div/div/div/div[8]/div/div/div/div/ul/li[1]/a[2]"
));

        JavascriptExecutor js = (JavascriptExecutor)
driver;
        js.executeScript("arguments[0].click();",
addToCartButton);

        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        WebElement goToCartButton =
driver.findElement(By

.xpath("//*[@id=\"masthead\"]/div[2]/div/div[3]/div[3]/div[
1]/a"));
```

```
            js = (JavascriptExecutor) driver;
            js.executeScript("arguments[0].click();",
goToCartButton);

      String expected = "Портмоне Pro-Covers ПКМ-40, черный";
      WebElement messageElement = driver.findElement(By
      .xpath("//*[@id=\"post-
5\"]/div/div/div/div/div/form/table/tbody/tr[1]/td[3]/a"));
            String messageText = messageElement.getText();
      Assertions.assertEquals(expected, messageText);
          }
```

Як можна побачити побачити на лістингу, у даній реалізаії ми використали технологію JavaScript Executor, яку не використовували раніше. JS Executor вказує, що драйвер може виконувати JavaScript, надаючи доступ до механізму для цього. Використати JS Executor нам довелось тому, що витягнути локатор для товару на сторінці було дуже важко через тонкощі написання коду сторінки, а JS Executor вирішив цю проблему.

Також ми використали Thread Sleep з таймером в 5000 мілісекунд для того, щоб сторінка змогла підгрузити всю інформацію. Так потрібно робити у тих випадках, коли тест виконується занадто швидко і інформація на веб-сторінці не встигає повністю відбразити всю інформацію.

У Appium також є інші способи очікування сторінок, такі як Implicit Wait та Explicit Wait:

Implicit wait — це спосіб повідомити драйверу Appium опитувати DOM (модель об’єкта документа) протягом певного часу, перш ніж викликати виняток, у якому він не може знайти елемент на сторінці. Значення часу очікування за замовчуванням встановлено на 0 секунд. Як тільки ми встановимо неявне очікування на певний час, воно зберігається протягом усього життя екзем-

пляра об’єкта webdriver. Встановлюється неявне очікування командою `appiumDriver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);`

Отже, це означає, що екземпляр драйвера може почекати максимум 10 секунд, перш ніж викликати виняток `NoSuchElement`. Нам потрібно бути уважними щодо неявного використання.

Шаблон Appium зазвичай дає нам код з неявною реалізацією очікування, тому зверніть увагу на попередній рядок у файлі класу `HomePageWebSteps`, як показано:

```
appiumDriver = new AppiumDriver (new URL(http://0.0.0.0:4723/wd/hub), capabilities);
appiumDriver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
```

Збільшення неявного тайм-ауту очікування слід використовувати з розумом, оскільки це матиме несприятливий вплив на загальний час виконання тесту, особливо якщо використовується з повільнішими стратегіями локатора, такими як `xpath`.

Це просто усуває багато недетермінованого очікування з коду. Неявне очікування найбільш підходить, коли час відповіді програми різниться через швидкість мережі.

Explicit wait, в свою чергу, можна описати таким чином:

Бувають випадки, коли програма, що тестується, може працювати повільно на певних елементах, як-от надсилання сторінки, надсилання форми або десь вона отримує дані із зовнішньої системи та займає трохи більше часу для завантаження. У цьому випадку використання неявного очікування для вирішення ситуації буде хибним підходом, враховуючи, що він повинен чекати для кожного елемента один і той же вказаний час.

Для вирішення цієї ситуації можливо використовувати явне очікування для таких елементів. У явному очікуванні ми повідомляємо екземпляру веб-драйвера чекати певної умови, викликаної через `ExpectedConditions`.

Отже, це очікування застосовується явно до зазначеного елемента. Явне очікування можна викликати за допомогою цього коду:

```
WebDriverWait wait = new WebDriverWait (appiumDriver, 10);
Wait.until (ExpectedConditions.visibilityOfElement-Located(By.id("text1")));
```

У попередньому коді ми створюємо екземпляр `WebDriverWait` з максимальним часом очікування 10 секунд, а потім використовуємо `ExpectedConditions`, який наказує драйверу зачекати, поки не буде виявлено видимість зазначеного елемента. `ExpectedConditions` має безліч методів, доступних для використання в різних умовах. `WebDriverWait` за замовчуванням викликає `ExpectedConditions` кожні 500 мс, поки не повернеться успішно, інакше він генерує виключення `TimeoutException`, як показано нижче:

```
org.openqa.selenium.TimeoutException: Expected condition failed: waiting for visibility of element located by By.id: text1 (tried for 10 second(s) with 500 MILLISECONDS interval)
```

Під час автоматизації вам, як правило, потрібно виконати задані умови для елемента, і для кожного з наступних умов ExpectedConditions надає набір попередньо визначених умов:

- Web element is present and clickable.
- Web element is selected.
- Web element is invisible.
- Selected web element.
- Presence of web element located by.
- Wait for a particular condition.
- Text present in a web element.

Список усіх методів, доступних у ExpectedConditions можна побачити на рисунку 20.

Явне очікування також використовується для перевірки вказаної властивості елемента, наприклад видимості, можливості натискання, невидимості та стану виділення [12].

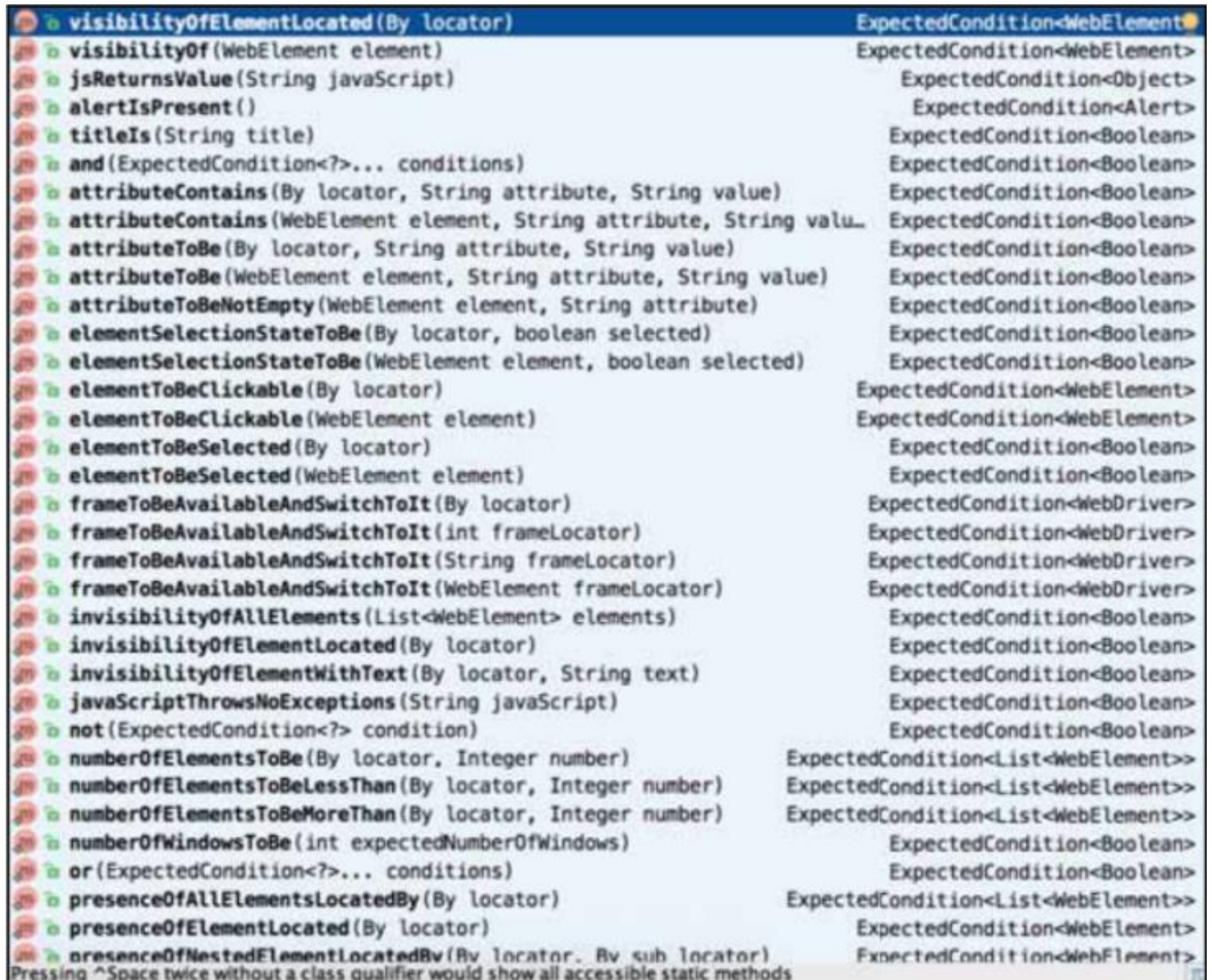


Рис. *20 Список методів ExpectedConditions*

Повертаючсь до тестового сценарію, що був описаний, очікуваним результатом даного тестового сценарію є те, що доданий товар відображається у кошику користувача (див. рис. 21). Тому для середовища розробки ми вказуємо, що очікуваний результат — це строкова назва доданого товару на сторінці кошику.

Як результат, покрокове виконання такого тестового сценарію виглядає таким чином:

- Переходимо до сайту, який будемо тестувати.

- Знаходимо необхідний товар.
- Клікаємо по кнопці додавання товару в кошик.
- Клікаємо по кнопці редіректу до кошика.
- Виконуємо порівняння фактичного результату з очікуваним.

Після того, як ми завершили реалізацію всіх тестових сценаріїв, потрібно розробити метод, що зупинить роботу Appium Driver. Цей метод дуже короткий, тому в окремий лістинг його виносити намає сенсу. Для того, щоб завершити роботу Appium Driver потрібно використати команду `public static void tearDown() { driver.quit(); }`.

Обов'язковим критерієм для запуска тестових сценаріїв є те, що для того щоб запустити наші тести нам необхідно спочатку підключити фізичний або емульований девайс до комп'ютера, після чого ініцалізувати Appium Server за допомогою командної строки, яку ми вводимо у CMD: `appium –chromedriver-executable /path/to/my/chromedriver.exe`. Ця команда запустить локальний Appium Sever на комп’ютері. Потім можно спокійно запускати наші тести. У випадку помилок, ми завжди можемо подивитись логи, які транслюються у Appium Server у вікні командної строки.

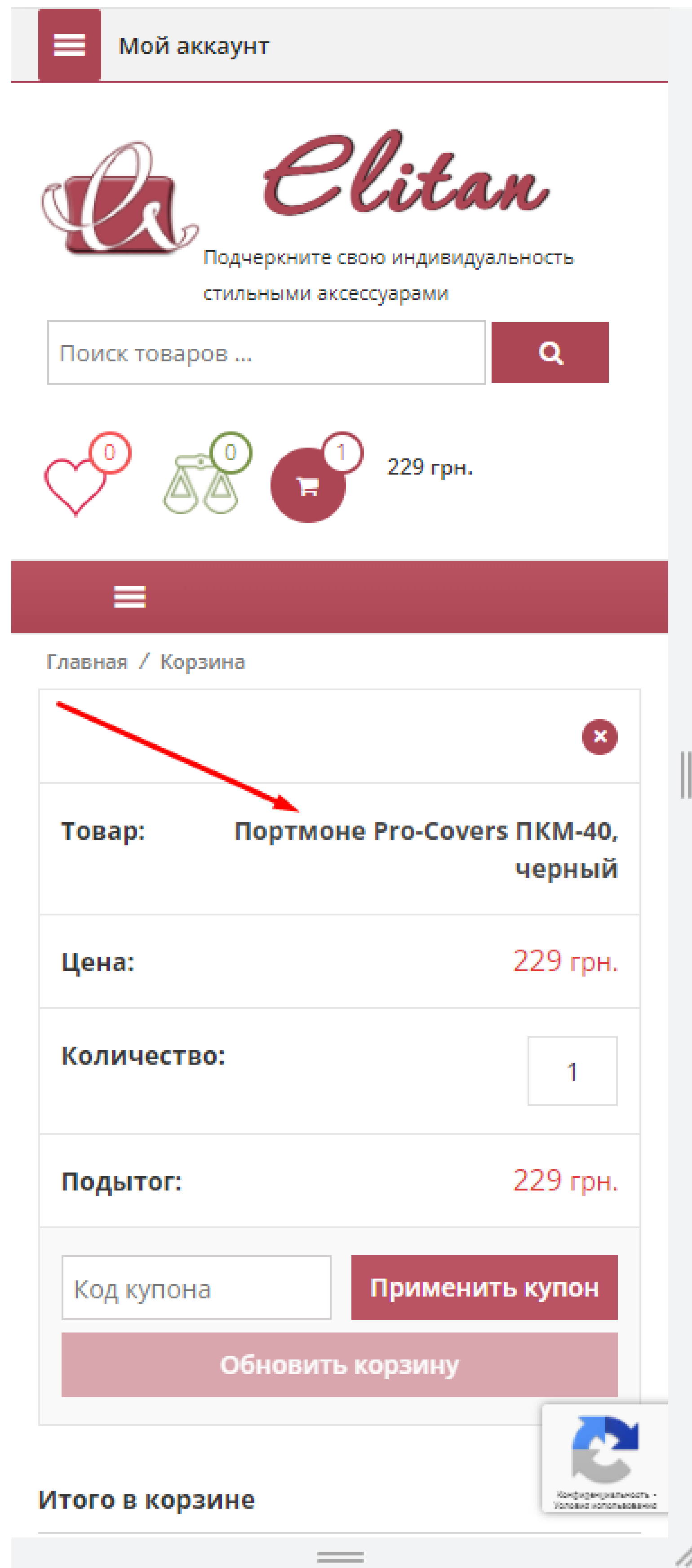


Рис. *21 Очікуваний результат тесту додавання товару до кошика*

3.2 Програмна реалізація тестування гібридного застосунку

Реалізація тестування гібридного застосунку досить не суттєво відрізняється від реалізації тестування веб-застосунку.

Для того, щоб реалізовувати тестові сценарії у гібридних застосунках нам, перш за все, необхідний сам застосунок, який ми встановлюємо на наш смартфон.

Після чого, знову-таки, підключаємо смартфон до комп'ютера або запускаємо емулятор з нашим пристроєм, на якому буде виконуватись тестування. Потім у командній строці вводимо вже знайому команду `appium –chromedriver-executable /path/to/my/chromedriver.exe` та розпочинаємо програмну реалізацію тестових сценаріїв.

Коли Appium сервер ініціалізовано, повертаємося до середовища розробки та створюємо клас у якому будемо виконувати тестування, назовемо його `ElitanHybriTest`.

У створеному класі ми знову створюємо змінну для AndroidDriver командою `protected static AndroidDriver<WebElement> driver;`

Тепер потрібно визначити для середовища розробки назву нашого мобільного застосунку, який буде тестуватись. Для цього використовуємо команду `protected static String appName = "ElitanMobileHybrid.apk";`

Тепер розпишемо команду що дозволить визначати тип девайсу, фізичний, чи емульований. Для цього ми використовуємо команду `private static final boolean isRealDevice = true;`

Після чого ми оголошуємо метод з поміткою `@BeforeAll` у якому ми розпишемо наші Desired Capabilities:

Лістинг *12 Налаштування Desired Capabilities у гібридному застосунку*

```
@BeforeAll
    public static void setUpClass() throws IOException {
```

```
            File classpathRoot = new
File(System.getProperty("user.dir"));
            File appDir = new File(classpathRoot, "apps");
            File app = new File(appDir.getCanonicalPath(),
appName);
            DesiredCapabilities capabilities = new
DesiredCapabilities();
            if (isRealDevice) {

capabilities.setCapability(MobileCapabilityType.DEVICE_NAME
, "device");

capabilities.setCapability(MobileCapabilityType.PLATFORM_VE
RSION, "9.0");
            } else {

capabilities.setCapability(MobileCapabilityType.DEVICE_NAME
, "Android10Phone");

capabilities.setCapability(MobileCapabilityType.PLATFORM_VE
RSION, "10.0");
            }

capabilities.setCapability(MobileCapabilityType.PLATFORM_NA
ME, "Android");

capabilities.setCapability(MobileCapabilityType.AUTOMATION_
NAME,
                    "UiAutomator2");
            capabilities.setCapability("app",
app.getAbsolutePath());
            capabilities.setCapability("autoWebview",
true);
```

```
            driver = new AndroidDriver<>(new
URL("http://127.0.0.1:4723/wd/hub"), capabilities);
            driver.manage().timeouts().implicitlyWait(30,
TimeUnit.SECONDS);
            /*Print contexts*/
            Set<String> contextNames =
driver.getContextHandles();
            for (String contextName : contextNames) {
                System.out.println(contextName);    //prints
out something like NATIVE_APP \n WEBVIEW_1
            }
        }
```

Після того, як ми виконали налаштування залежностей, ми можемо виконувати реалізацію тестів. Для цього нам знову необхідно подати сигнал для середовища розробки, що наступний метод буде методом, що зберігає в собі тест. Для цього ми використовуємо команду @Test.

Далі опишемо тест, який пройде авторизацію на сайті з використанням валідних даних.

Лістинг *13 Опис тесту авторизації у гібридному застосунку з використанням валідних даних*

```
        @Test
        void testLogin() {
            /*Registered user login and password*/
            String email = "usertest@gmail.com";
            String password = "testtest";
            String username = "Testirovshichik";

            driver.get("https://elitan.com.ua/");
            driver.findElement(By.linkText("Вход/ Регистра-
ция")).click();
```

```
            /*Opens authorization page /my-account*/

driver.findElement(By.cssSelector("#username")).sendKeys(em
ail);

driver.findElement(By.cssSelector("#password")).sendKeys(pa
ssword);

driver.findElement(By.cssSelector(".woocommerce-form-
login__submit")).click();

            String expected = "Добро пожаловать, " + username
+ " (не " + username + "? Выйти)";
            WebElement messageElement = driver.findElement(
                    By.cssSelector(".woocommerce-MyAccount-
content > p:nth-child(2)"));
            Assertions.assertEquals(expected,
messageElement.getText());
        }
```

Як ми можемо побачити, у гібридному застосунку використовуються ті ж самі локатори, що й для веб-застосунку. Зумовлено це тим, що гібридний застосунок використовує технологію WebView, яка відповідає за відкриття веб-сторінок у мобільних застосунках.

Відповідно, для того, щоб виконати авторизацію на сайті з використанням валідних даних, ми знову оголошуємо строкові змінні в які передаємо інформацію, що будемо використовувати для заповнення полів.

Після чого за допомогою команди

```
driver.get("https://elitan.com.ua/");
```

ми вказуємо веб-драйверу, що нам необхідно перейди до веб-сайту для тестування, потім ми переходимо до сторінки авторизації за допомогою команди

```
driver.findElement(By.linkText("Вход/ Регистра-
ция")).click();
```

Коли ми потрапили на сторінку авторизації за допомогою локаторів ми визначаємо поля, які необхідно заповнити даними та посилаємо в них значення, які зберігаються у строкових змінних, які ми ініціалізували раніше.

Після того як ми заповнили дані, посилаємо команду веб-драйверу, яка клікне по кнопці для входу за допомогою команди

driver.findElement(By.cssSelector(".woocommerce-form-login__submit")).click();

Після чого необхідно порівняти фактичний результат з очікуваним. Реалізація порівняння результатів у гібридному застосунку не відрізняється від порівняння результатів у веб-застосунку, тому можна використати код, що ми писали раніше.

Після порівняння фактичного результату з очікуваним ми можемо створювати нові тести за тим же способом, як і перший описаний тест. Для цього потрібно лише зробити анотацію `@Test`, а потом описувати метод, який буде виконувати тестування.

Обов'язково, також як і у веб-застосунку необхідно наприкінці тестування завершити роботу Appium Driver. Для цього необхідно подати сигнал про те, що ми плануємо завершити роботу драйверу за допомогою команди `@AfterAll` та описати коротенький метод `@AfterAll public static void tearDown() { driver.quit(); }` для того, щоб Appium Driver успішно завершив свою роботу після виконання всіх тестів.

3.3 Програмна реалізація тестування нативного застосунку

Реалізація тестування нативного застосунку досить суттєво відрізняється від тестування веб та гібридних застосунків. Це зумовлено тим, що у нативних застосунків вже по іншому виконується пошук локаторів, та відпадає необхідність у використанні ChromeDriver.

Для того, щоб розпочати тестування нативних застосунків перш за все необхідно під'єднати мобільний девайс до комп'ютера, після чого потрібно запустити Appium Server. Для цього в командній строці ми вписуємо команду `appium` та трохи очікуємо повної ініціалізації серверу.

Після чого повертаємося до середовища розробки та прописуємо команду для оголошення відомого вже нам AndroidDriver: `protected static AndroidDriver<WebElement> driver;`

Потім обов'язково оголошуємо строкову змінну, що буде зберігати у собі назву нашого нативного застосунку. Для цього використовуємо команду `protected static String appName = "ElitanMobileNative.apk";`

Також використовуємо команду для визначення типу мобільного девайсу, на якому буде проводитись тестування, а саме фізичного чи емульованого, за допомогою команди `private static final boolean isRealDevice = false;`

Після чого ми оголошуємо метод з поміткою `@BeforeAll` у якому ми розпишемо наші Desired Capabilities:

Лістинг *14 Налаштування Desired Capabilities у нативному застосунку*

```
@BeforeAll
    public static void setUpClass() throws IOException {
        File classpathRoot = new File(System.getProperty("user.dir"));
        File appDir = new File(classpathRoot, "apps");
        File app = new File(appDir.getCanonicalPath(), appName);

        DesiredCapabilities capabilities = new DesiredCapabilities();
        if (isRealDevice) {
```

```
capabilities.setCapability(MobileCapabilityType.DEVICE_NAME
, "device");

capabilities.setCapability(MobileCapabilityType.PLATFORM_VE
RSION, "9.0");
            } else {

capabilities.setCapability(MobileCapabilityType.DEVICE_NAME
, "Android10Phone");

capabilities.setCapability(MobileCapabilityType.PLATFORM_VE
RSION, "10.0");
            }

capabilities.setCapability(MobileCapabilityType.PLATFORM_NA
ME, "Android");

capabilities.setCapability(MobileCapabilityType.AUTOMATION_
NAME,
                    "UiAutomator2");
            capabilities.setCapability("app",
app.getAbsolutePath());
            driver      =      new      AndroidDriver<>(new
URL("http://127.0.0.1:4723/wd/hub"), capabilities);
            driver.manage().timeouts().implicitlyWait(30,
TimeUnit.SECONDS);
        }
```

Як ми можемо побачити, ми не використовуємо у залежностях appName чи packageName. У нашому випадку ми користуємося методом getCanonicalPath для того, щоб отримати правильне ім’я необхідного нам для тестування

застосунку. Для того, щоб було зрозуміло, getCanonicalPath повертає канонічний рядок імені шляху абстрактного імені шляху.

Канонічний шлях є абсолютним і унікальним. Точне визначення канонічної форми залежить від системи. Цей метод спочатку перетворює це ім’я шляху в абсолютну форму, якщо це необхідно, ніби викликаючи метод getAbsolutePath, а потім відображає його в його унікальну форму залежно від системи. Зазвичай це передбачає видалення зайвих імен, таких як "." і ".." із імені шляху, розв'язування символічних посилань (на платформах UNIX) та перетворення літер дисків у стандартний регістр (на платформах Microsoft Windows).

Кожне ім’я шляху, що позначає існуючий файл або каталог, має унікальну канонічну форму. Кожне ім’я шляху, що позначає неіснуючий файл або каталог, також має унікальну канонічну форму. Канонічна форма імені шляху до неіснуючого файлу або каталогу може відрізнятися від канонічної форми того самого імені шляху після створення файлу або каталогу. Аналогічно, канонічна форма імені шляху до існуючого файлу або каталогу може відрізнятися від канонічної форми того самого імені шляху після видалення файлу або каталогу.

Після того, як залежності налаштовані та AndroidDriver ініціалізовано, можна реалізовувати тестові методи.

Тестові методи ініціалізуються за тим же шляхом, що й у гібридних та веб-застосунках, за допомогою сигналу `@Test`.

Далі на лістингу 9 буде продемонстровано тест авторизації у нативному застосунку за допомогою валідних даних.

Лістинг 15 *Реалізація тесту авторизації у нативному додатку з використанням валідних даних*

```
@Test
    void testLogin() {
        /*Registered user login and password*/
```

```
            String email = "usertest@gmail.com";
            String password = "testtest";


driver.findElementById("ua.com.elitan.elitanmobilenative:id
/emailText").sendKeys(email);

driver.findElementById("ua.com.elitan.elitanmobilenative:id
/passwordText").sendKeys(password);

driver.findElementById("ua.com.elitan.elitanmobilenative:id
/loginButton").click();

            String expected = "Добро пожаловать";
            MobileElement messageElement = (MobileElement)
driver

.findElementById("ua.com.elitan.elitanmobilenative:id/messa
geText");
            Assertions.assertEquals(expected,
messageElement.getText());
        }
```

Як можна побачити, суттєвою відмінністю від реалізації тестів у гібридних та веб-застосунках є те, що у нативних застосунках відрізняються локатори. Тут ми вже не зможем використувати пошук елементів по xPath чи CssSelector, бо ці локатори відносять до веб сторінок. У нашому випадку ми використовуємо пошук елементів по їх ID.

Спочатку визначаються поля «Email» та «Password», в які передаються дані, що оголошені у строкових змінних, після чого натискається кнопка для входу у мобільний застосунок, та відображається інформаційне повідомлення «Добро пожаловать!».

Для того, щоб виконати порівняння фактичного результату з очікуваним, ми знов оголошуємо очікувану строкову змінну, в яку передаємо очікуване значення після тесту. У нашому випадку очікуваним результатом і буде повідомлення «Добро пожаловать!», для цього використовуємо команду String expected = "Добро пожаловать";

Після чого просто шукаємо через пошук локаторів це саме повідомлення з однією відмінністю — ми шукаємо елемент з інформаційним повідомленням по ID.

Обов’язково наприкінці тестів потрібно завершити роботу Appium Driver. Для цього використовуємо метод, який ми використовували у веб та гібридних застосунках: `@AfterAll public static void tearDown() { driver.quit(); }`

РОЗДІЛ 4 РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

4.1 Чому все ж таки Appium

Під час написання дипломної роботи я проводив досить багато досліджень. Мною була зібрана інформація про різні фреймворки для тестування мобільних, веб та нативних застосунків, я пробував використовувати декілька різних інструментів для пошуку локаторів у тих самих застосунках та тепер можу зробити підсумок.

Appium — як інструмент для тестування мобільних застосунків є найбільш унікальним та найбільш простим рішенням, якщо необхідно провести автоматизоване тестування будь-якого застосунку. Цей фреймворк дозволяє не тільки тестувати веб-сайти, він повністю підтримує також тестування нативних та гібридних застосунків, чого не можуть собі дозволити більшість інших фреймворків.

Appium — це Open-Source фреймворк. Цей факт вже робить наголошення на тому, що він буде популярним та його будуть використовувати. Є також фреймворки, які є безкоштовними для використання, такі як Espresso, Robotium чи Selendroid, але вони не йдуть у порівняння з можливостями фреймворку Appium, бо Robotium, наприклад, не підтримує тестування веб-застосунків, а Espresso, також наприклад, підтримує тільки моу розробки Java.

Appium має свій власний інспектор, тобто інструмент для пошуку локаторів, а це значно облегшує роботу користувачеві, який буде писати тести для нативного чи гібридного застосунку. Цей інструмент постійно покращується розробниками, що робить його наразі найбільш перспективним інструментом для пошуку локаторів.

4.2 Аналіз результатів дослідження

Під час роботи з Appium не все було так ідеально, як хотілось. Не винятками стали нюанси, які виникали. Коли тестується веб-застосунок, основним нюансом є те, що необхідно використовувати один веб-драйвер, який має мати таку ж версію, як і версія браузеру у смартфоні. Це означає те, що коли на вашому смартфоні виходить оновлення веб-браузеру, то необхідно перевірити, до якої версії він оновився та, відповідно, завантажити та підключати вже новий веб-драйвер. На щастя, усі веб-драйвери оновлюються миттево, тому лише потрібно бути уважним до програмних оновлень.

Також в результаті досліджень, виконаних під час роботи над дипломним проектом було визначено деякі плюси та мінуси тестування веб та нативних застосунків.

Плюси тестування веб-застосунків:

- Легко знаходити локатори за допомогою реального веб-браузера.
- Можна використовувати Appium Inspector або Selenium IDE для запису тестів.

Мінуси тестування веб-застосунків:

- Інколи локатори бувають не точними через використання Appium інспектору.

Плюси тестування нативних застосунків:

- Локатори на Android можна знаходити через UIautomatorviewer та Appium Inspector.
- Можна використовувати ту ж мову розмітки, що й для веб-застосунків.

Мінуси тестування нативних застосунків:

- Важко знаходити локатори через те, що додається більше неопрацьованих виключень для за стосунку.
- Важче тестувати через те, що є риски непередбачуваного завершення роботи застосунку.
- Важче тестувати через те, що деякі локатори важко знайти.

ВИСНОВКИ

1. Виконане дослідження сучасних фреймворків автоматизації тестування мобільних застосунків, за його результатами було обрабно для подальшого дослідження фреймворк Appium, як найбільш перспективний Open-Source фреймворк.

2. У роботі було виконано розгортання середовища, створення та виконання тестів у фреймворку Appium та виконані налаштування бажаних можливостей для тестування на емуляторі та фізичному пристрої.

3. Для веб, гібридного та нативного застосунку роботи з інтернет-магазином були спроектовані тестові сценарії для входу та реєстрації користувача.

4. За результатами аналізу були виявлені недоліки роботи Appium Inspector при визначенні локаторів веб та гібридного застосунку. Тому було запропоновано використовувати для цього Selenium IDE з записом дій користувача відповідно тестовому сценарію та подальшим експортом тестового скрипту у код тесту на мові Java.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Verma Nishant. Mobile Test Automation with Appium 2017. 191 с. ISBN-13 978-1787280168.

2. Офіційний репозиторій Appium https://github.com/appium/appium-xcuitest-driver

3. Налаштування Desired Capabilities в Appium https://appium.io/docs/en/writing-running-appium/caps/

4. Історія розвитку тестування http://csaa.ru/istorija-razvitija-testirovanija-programmnogo/

5. What is Test Automation? By Eran Kinsbrunner https://www.perfecto.io/blog/what-is-test-automation

6. Автоматизация тестирования: учимся экономить. Автор: Владисалв Орликов. https://software-testing.ru/library/around-testing/processes/437-learn-to-save

7. Автоматизація тестування: http://www.williamspublishing.com/PDF/978-5-8459-1625-9/part.pdf

8. Порівняння інструментів автоматизації https://docs.google.com/spreadsheets/d/1gfPEakdSgizwokM-34B1P-16MRqmryM8/edit#gid=1246484834

9. Endpoints у веб-драйвері https://w3c.github.io/webdriver/#endpoints

10. Appium Essentials https://www.amazon.com/Appium-Essentials-Manoj-Hans/dp/1784392480

11. Appium Wiki https://en.wikipedia.org/wiki/Appium

12. Офіційний веб-сайт Appium https://appium.io/

13. Введення в Appium https://coderlessons.com/tutorials/kachestvo-programmnogo-obespecheniia/mobilnoe-testirovanie/3-vvedenie-v-appium

14. Тестування мобільних застосунків з Appium https://www.a1qa.ru/blog/avtomatizatsiya-testirovaniya-mobilnyh-prilozhenij-s-appium/

15. Preparing Appium Tests for Upload https://docs.microsoft.com/en-us/appcenter/test-cloud/frameworks/appium/

16. Мобильная автоматизация с appium – опыт написания первого теста https://automated-testing.info/t/mobilnaya-avtomatizacziya-s-appium-opyt-napisaniya-pervogo-testa/17221

17. Почему Appium для народа? https://habr.com/ru/post/488482/

18. Running Tests https://appium.io/docs/en/writing-running-appium/running-tests/

19. Prepating your app for test https://appium.io/docs/en/writing-running-appium/running-tests/#preparing-your-app-for-test-android

20. Appium Native Java Tutorial https://applitools.com/tutorials/appium-native-java.html#install-the-sdk

21. Руководство по Appium для тестирования устройств под Android и iOS http://getbug.ru/rukovodstvo-po-appium-dlya-testirovaniya-ustroystv-pod-andoid-i-ios/

22. Appium Tutorial Step by Step Appium Automation by Onur Baskirt https://www.swtestacademy.com/appium-tutorial/

23. How to do Appium setup with Testing in Android Studio? https://www.qss-technosoft.com/appium-setup-with-testing-in-android-studio

24. Инструкция по установке ПО, необходимого для тестирования Android приложений https://jazzteam.org/ru/technical-articles/instructions-for-installing-software-needed-to-test-android-applications/

25. Appium-adb https://www.npmjs.com/package/appium-adb

26. Android Debug Bridge (adb) https://developer.android.com/studio/command-line/adb

27. How to Install ADB on Windows, macOS and Linux https://www.xda-developers.com/install-adb-windows-macos-linux/

28. Команды ADB для Android пользователей https://losst.ru/komandy-adb-dlya-android-polzovatelej

29. IEEE Guide to Software Engineering Body of Knowledge, SWEBOK, 2004

Декларація

академічної доброчесності

здобувача ступеня вищої освіти ЗНУ

Я, Ревякін Дмитро Сергійович, студент _2_ курсу, форми навчання денної, Інженерного навчального-наукового інституту, спеціальність 121 Інженерія програмного забезпечення, адерса електронної пошти sp151bd-24@stu.zsea.edu.ua, — підтверджую, що написана мною кваліфікаційна робота на тему **«Appium як сучасний інтрумент автоматизації мобільних застосунків»** відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст. 42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірка електронна версія роботи є ідентичною її друкованій версії;

згоден/згодна на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-систем, а також на архівування моєї роботи в базі даниї цієї системи.

Дата 07.12.2021 Підпис ______________ Ревякін Дмитро Сергійович

(студент)

Дата 07.12.2021 Підпис ______________ Коломоєць Генадій Павлович

(науковий керівник)

Декларація

академічної доброчесності

здобувача ступеня вищої освіти ЗНУ

Я, Ревякін Дмитро Сергійович, студент _2_ курсу, форми навчання денної, Інженерного навчального-наукового інституту, спеціальність 121 Інженерія програмного забезпечення, адерса електронної пошти sp151bd-24@stu.zsea.edu.ua, — підтверджую, що написана мною кваліфікаційна робота на тему **«Appium як сучасний інтрумент автоматизації мобільних застосунків»** відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст. 42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірка електронна версія роботи є ідентичною її друкованій версії;

згоден/згодна на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-систем, а також на архівування моєї роботи в базі даниї цієї системи.

Дата 07.12.2021 Підпис ______________ Ревякін Дмитро Сергійович
(студент)

Дата 07.12.2021 Підпис ______________ Коломоєць Генадій Павлович
(науковий керівник)