Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

Бакалаврська дипломна робота на тему:

«РОЗРОБКА ФРЕЙМВОРКУ ДЛЯ ТЕСТУВАННЯ МОБІЛЬНОГО ДОДАТКУ JETIQ»

Виконав: студент 4 курсу групи 2ПІ–18б
спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Олійник І. А.
(прізвище та ініціали)

Керівник: к. т. н., доц. Коваль Л. Г.
(прізвище та ініціали)
«____» _____ 2022 р.

Рецензент: д. т. н., проф. каф. КН Іванчук Я. В.
(прізвище та ініціали)
«____» _____ 2022 р.

Допущено до захисту
Зав. кафедри ПЗ Романюк О. Н. _____
«____» _____ 2022 р.

Вінниця ВНТУ– 2022 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
25 березня 2022 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ
Олійнику Івану Андрійовичу

1. Тема роботи: Розробка фреймворку для тестування мобільного додатку JetIQ, керівник роботи: Коваль Л. Г., к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від "24" березня 2022 року № 66.

2. Строк подання студентом роботи 13 червня 2022 року.

3. Вихідні дані до роботи: модель розробки – ітеративна; вхідні дані – описані сценарії тестування, налаштоване середовище емуляції мобільного додатку; вихідні дані – логи виконання тестування; середовище розробки – IntelliJ IDEA; мова програмування – JAVA.

4. Зміст розрахунково-пояснювальної записки: вступ; обґрунтування вибору типу тестування та підходів до розробки фреймворку для тестування мобільного додатку; аналіз технологій, підходів та систем для реалізації автоматизованого тестування; розробка фреймворку для тестування мобільного додатку jetiq; тестування мобільного додатку jetiq за допомогою розробленого фреймворку; висновки; список використаних джерел; додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): титульний слайд; цілі дослідження; впровадження розробки; діаграма класів; блок-схема створення тестів; середовище та мова розробки; методологія розробки; середовище емуляції android пристрою; підсумки; фінальний слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Коваль Л. Г., к.т.н., доцент кафедри ПЗ		

7. Дата видачі завдання 25 березня 2022 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту	Термін виконання етапів проекту (роботи)	Примітка
1	Аналіз предметної області	26.03.22 – 08.04.22	Виконано
2	Аналіз сучасного стану автоматизованого тестування	11.04.22 – 21.04.22	Виконано
3	Розробка архітектури програми	22.04.22– 30.04.22	Виконано
4	Програмна реалізація додатку	02.05.22– 13.05.22	Виконано
5	Тестування роботи додатку	16.05.22– 24.05.22	Виконано
6	Оформлення матеріалів до захисту БДР	27.05.22– 13.06.22	Виконано

Студент _____ **Олійник І. А.**
(підпис) (прізвище та ініціали)

Керівник бакалаврської дипломної роботи _____ **Коваль Л. Г.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 72 сторінок формату А4, на яких є 31 рисунків, список використаних джерел містить 13 найменувань.

Метою бакалаврської дипломної роботи є оптимізація автоматизації тестування мобільного додатку JetIQ.

У бакалаврській дипломній роботі розроблено фреймворк для тестування мобільного додатку JetIQ. Розробка фреймворку виконана мовою програмування JAVA у середовищі програмування IntelliJ IDEA. Фреймворк дозволяє виконувати автоматизоване тестування мобільного додатку JetIQ.

Ключові слова: BDD-методологія, Gherkin, фреймворк, мобільний додаток, тестування.

ANNOTATION

The bachelor's thesis consists of 72 A4 pages, which have 31 figures, a list of sources contains 13 items.

The aim of the bachelor's thesis is to optimize the automation of testing the JetIQ mobile application.

A framework for testing the JetIQ mobile application has been developed in the bachelor's thesis. The framework was developed in the JAVA programming language in the IntelliJ IDEA programming environment. The framework allows you to perform automated testing of the JetIQ mobile application.

Keywords: BDD-methodology, Gherkin, framework, mobile application, testing.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. На сьогоднішній день метою людства є налаштування процесів, що необхідні для життєдіяльності на рівень автоматичного виконання. Ця ціль далеко від впровадження у реальності. Проте і на даний момент людство йде у напрямку видалення людського фактору із буденних рутинних процесів. На кшталт онлайн замовлень у закладах, що виключає шанс того що ваше замовлення перетвориться у «зіпсований телефон», адже у такому разі прибирається шанс що офіціант помилиться у записі вашого замовлення.

Ця мета не пройшла і повз розробку програмного забезпечення, у процесі якої для тестування використовують тестувальників, що виконують величезну кількість одноманітної роботи, за умови відсутності автоматизованого тестування.

Автоматизоване тестування економить для замовника кошти, а також переймає на себе величезний частину роботи, чим сильно заощаджує час, у який входять окрім перевірки функціоналу: багаторазові перевірки стандартних функцій таких як пошук або логін, постійна перевірка програмного забезпечення та внесених змін, виконання регресії, що в собі містить перевірку чималої кількості тестів. І усе це займе величезну кількість часу при виконанні вручну. Таким чином, впровадивши автоматизоване тестування сильно зменшується кількість людей що відповідають за рутинну роботу, адже їх лише потрібно слідкувати за якістю тестів та оновлювати їх та надаємо змогу виконувати вручну більш важливі задачі.

Автоматизація особливо підходить для регресивного тестування (яке перевіряє, чи все добре після змін), тестування на різних платформах і з різними конфігураціями, функціональне тестування, тестування з навантаженням, стрес-тестування, мобільного тестування, тощо.

Саме тому розробка фреймворку для тестування є актуальною на сьогодні.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою бакалаврської дипломної роботи є оптимізація автоматизованого тестування мобільного додатку JetIQ.

Основними задачами дослідження є:

- обґрунтування вибору типу тестування та підходів до розробки фреймворку для тестування мобільного додатку;
- аналіз технологій, підходів та систем для реалізації автоматизованого тестування;
- розробка фреймворку для тестування мобільного додатку JetIQ;
- проведення тестування мобільного додатку JetIQ за допомогою розробленого фреймворку.

Об’єкт дослідження – процес автоматизованого тестування мобільного додатку.

Предмет дослідження – методи та засоби розробки фрейморків для автоматизованого тестування.

Методи дослідження. У процесі досліджень використовувались: теорія алгоритмів, теорія програмування, методи тестування програмного забезпечення для розробки моделей та методів автоматизації тестування; комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Новизна отриманих результатів.

1. Вперше запропоновано метод автоматизації тестування мобільного додатку JetIQ, особливість якого полягає у використанні фреймворку, що дозволяє підвищити швидкість тестування мобільного додатку JetIQ в 5-10 разів у порівнянні з мануальним тестуванням.
2. Подальшого розвитку отримав метод розробки фреймворку для тестування мобільного додатку JetIQ, у якому, на відміну від існуючих,

використано BDD-методологію розробки програмного забезпечення, що спростило розробку тестів за рахунок побудови сценаріїв поведінки.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в бакалаврській дипломній роботі теоретичних положень запропоновано алгоритми та розроблено фреймворк для тестування мобільного додатку JetIQ.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській дипломній роботі, отримані автором особисто. У праці, опублікованій у співавторстві, автору належать такі результати: розробка моделі та методів тестування мобільного додатку JetIQ [1].

Апробація матеріалів бакалаврської дипломної роботи. Основні положення бакалаврської дипломної роботи доповідалися та обговорювалися на науково-технічній конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ) (Вінниця, 2022 р.).

Публікації. Основні результати досліджень опубліковано в одній статті у матеріалах конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ) (Вінниця, 2022 р.).

1 ОБҐРУНТУВАННЯ ВИБОРУ ТИПУ ТЕСТУВАННЯ ТА ПІДХОДІВ ДО РОЗРОБКИ ФРЕЙМВОРКУ ДЛЯ ТЕСТУВАННЯ МОБІЛЬНОГО ДОДАТКУ

1.1 Аналіз предметної області і сучасного стану питання

Тестування автоматизації можна визначити як спосіб повторного запуску набору тестів без необхідності виконувати їх вручну. Введення тестів автоматизації у вашу стратегію тестування – це спосіб заощадити гроші та час.

На даний момент автоматизоване тестування задіяне у чималій кількості проектів що розробляють програмне забезпечення. Проте ні на одному проекті немає такого фреймворку щоб він був ідеальним.

Від проекту до проекту вигляд та підхід до розробки фреймворків для автоматизованого тестування відрізняється, а також розвивається із кожним новим проектом та потоком тестувальників між проектами. Це сприяє розвитку нових можливостей та підходів до автоматизованого тестування.

Процес покращення фреймворків полягає у розробці універсальних методів та високому рівні оптимізації проекту, що впливає на стабільність та швидкість виконання тестів. А також тій множині різноманітного тестування необхідного для програмного забезпечення, яке команда може покрити.

Перехід різних галузей промисловості до мобільних робить мобільний сегмент одним з найбільш перспективних серед споживчих у IT-сфері.

Через швидкий розвиток ринку мобільного ринку не можна сказати, що він повністю готовий підтримувати всі складні архітектурні рішення, які вже давно існують на більш традиційних платформах. Є складнощі з оптимізацією та гарантією коректної роботи на різних пристроях. Проблема автоматизованого тестування стає як ніколи актуальною.

Наразі на ринку мобільних застосувань продовжують з’являтися усе нові технології та усе нові пристрої, що ускладнює процес тестування та збільшує кількість необхідних тестів.

Тож саме розвиток науки у цих галузях має допомогти розв’язати проблему необхідності тестувати все більше різного коду.

1.2 Типи тестування

У залежності від переслідуваних цілей, види тестування можна умовно розділити на наступні типи:

- функціональне тестування (functional);
- нефункціональне тестування (non-functional);
- тестування пов’язане зі змінами.

За знанням системи:

- тестування чорної скриньки (black box);
- тестування білої скриньки (white box);
- тестування сірої скриньки (gray box).

За ступенем автоматизації:

- ручне тестування (manual testing);
- автоматизоване тестування (automated testing);
- напівавтоматизоване тестування (semiautomated testing).

Типи тестів визначають, який тип наборів тестів можна автоматизувати.

Ознайомимось із деякими видами тестування:

1. Функціональне тестування – написання тестів для перевірки ділової логіки та функціональних можливостей, які очікуються від програми.
2. Нефункціональне тестування – тести для перевірки вимог, що стосуються продуктивності, безпеки, баз даних тощо.
3. Модульне тестування – тести проводяться під час самої фази розробки, в ідеалі розробник після завершення розробки та перед передачею системи тестувальникам на тестування.
4. API тестування – тестування на основі запиту та відповіді, на яких побудована програма.
5. UI тестування – перевіряє функціональність та ділову логіку програми від лиця користувача та націлені на тестування функціональних можливостей або просто на тестування елементів інтерфейсу програми.

6. Unit Tests – це тести, які побудовані для тестування коду програми і зазвичай пишуться розробниками. Вони націлені на стандарти кодування, такі як написання методів та функцій[2].

Найчастіше у автоматизації тестування стикаємось саме із API та UI тестуванням.

API – це абревіатура від Application Programming Interface. API забезпечує зв'язок між двома окремими програмними системами.

Тестування API – це перевірка API програмної системи. Щоб перевірити API, його потрібно викликати з програмного забезпечення, що викликає. Перед тестуванням API необхідно підготувати необхідне тестове середовище, базу даних, сервер, програму.

Тестування API зосереджено на функціональності бізнес-логіки (наприклад, обчислення загальної ціни) і воно повністю відрізняється від тестування UI. В основному він зосереджений на рівні бізнес-логіки архітектури програмного забезпечення. Це тестування не буде зосереджено на зовнішньому вигляді програми.

UI розшифровується як User Interface. Інтерфейс користувача дозволяє користувачеві взаємодіяти з програмою.

Тестування інтерфейсу користувача відноситься до тестування графічних інтерфейсів користувача, наприклад, як користувач взаємодіє з програмою, тестування елементів програми, таких як шрифти, макети, кнопки, зображення, кольори тощо. В основному, тестування інтерфейсу користувача зосереджується на зовнішньому вигляді програми.

Для фреймворку було обрано тестування UI так як воно краще перевірить функціонал мобільного додатку та буде більш наглядним.

1.3 Підходи до розробки

Підходи до розробки поділяються за складністю, областями застосування та цілями.

TDD – Test Driven Development

TDD – це методологія розробки ПЗ, яка ґрунтується на повторенні коротких циклів розробки: спочатку пишеться тест, що покриває бажану зміну, потім пишеться програмний код, який реалізує бажану поведінку системи та дозволить пройти написаний тест. Потім проводиться рефакторинг написаного коду із постійною перевіркою проходження тестів.

BDD – Behavior Driven Development

BDD – це технологія, заснована на описі поведінки. Певна людина пише опис виду "я як користувач хочу коли натиснули кнопку пуск тоді показувалося меню як на картинці", для цього є спеціально виділені ключові слова. Given – задання контексту, When – виконання дій, Then – перевірка результату.

DDD – Domain Driven Design

DDD – це набір правил, що дозволяють приймати правильні проектні рішення. Цей підхід дозволяє значно прискорити процес проектування програмного забезпечення у незнайомій предметній галузі. Процес розробки зводиться створення програмних абстракцій, які називаються моделями предметних областей. У ці моделі входить бізнес-логіка, що встановлює зв'язок між реальними умовами галузі застосування продукту та кодом.

Підхід DDD особливо корисний у ситуаціях, коли розробник не є фахівцем у галузі продукту, що розробляється.

FDD – Features Driven Development

FDD є спробою об'єднати найбільш визнані в індустрії розробки програмного забезпечення методики, що приймають за основу важливу для замовника функціональність програмного забезпечення, що розробляється. Основною метою даної методології є розробка реального, працюючого програмного забезпечення систематично, у поставлені терміни.

Згідно з FDD, одна ітерація триває два тижні. FDD налічує п'ять процесів. Перші три з них належать до початку проекту. Останні два кроки потрібно робити під час кожної ітерації. У цьому кожен процес розбивається завдання і має критерії верифікації.

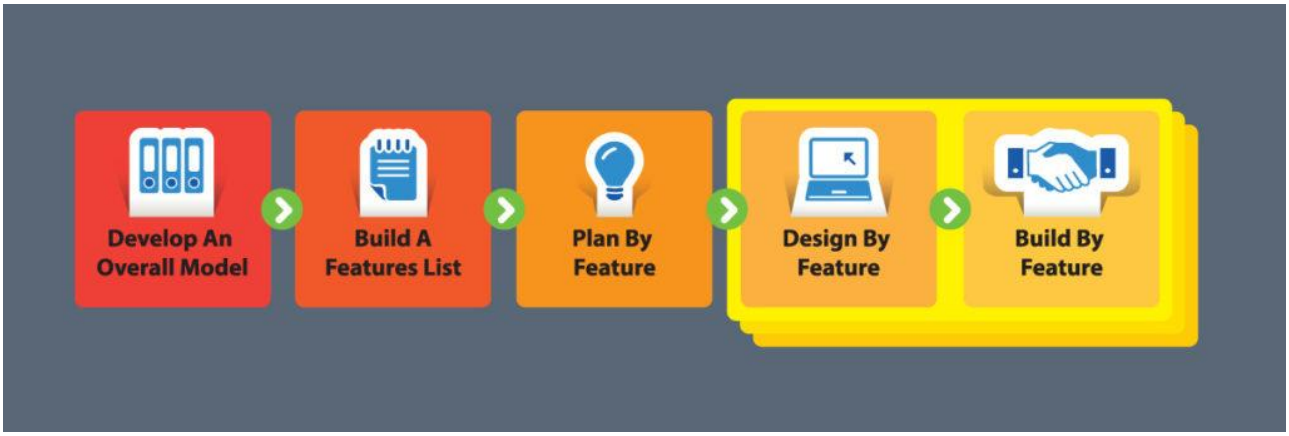

Рисунок 1.1 – Процеси FDD

Проаналізувавши підходи розробки для фреймворку було обрано Behavior Driven Development підхід.

MDD – це стиль розробки програмного забезпечення, де моделі стають основними артефактами розробки, з яких генерується код та інші артефакти.

Основна мета MDD – мінімізувати витрати, пов’язані з прив’язкою до певних системних платформ та програмної інфраструктури. Адже основна бізнес-логіка міститься в діаграмах і не обмежує нас вибором мови програмування та інструментів розробки.

Ідея MDD не нова – вона з різним успіхом використовувалася раніше. Причина підвищеної уваги до них в даний час полягає в тому, що набагато більше процесів піддається автоматизації, ніж раніше.

PDD – Panic Driven Development

PDD – це техніка, що стрімко підвищує швидкість роботи в команді будь-якого проекту за найкоротші терміни.

Вона знаходить застосування в компаніях по всьому світу та є фундаментом для гнучкого та безкомпромісного програмування. Ця техніка поділяє основні принципи методології розробки Agile, але вона позбавлена марних церемоній та технологічного навантаження, які лише знижують швидкість роботи команди.

Щойно серед спринту виникає нове завдання – її пріоритет здіймається над усією раніше запланованою роботою. Адже все нове завжди краще та важливіше[3].

1.4 Висновки

1. Для фреймворку обрано тестування UI, оскільки воно забезпечує перевірку функціоналу мобільного додатку та є більш наглядним у порівнянні з API-тестуванням.
2. Аналіз методів розробки фреймворків для тестування мобільних додатків показав, що переваги має BDD-методологія розробки програмного забезпечення, яка спрощує розробку тестів за рахунок побудови сценаріїв поведінки.

2 АНАЛІЗ ТЕХНОЛОГІЙ, ПІДХОДІВ ТА СИСТЕМ ДЛЯ РЕАЛІЗАЦІЇ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ

2.1 Аналіз системи тестування

Тестування програмного забезпечення – техніка контролю якості, що перевіряє відповідність між реальною і очікуваною поведінкою програми завдяки кінцевому набору тестів, які обираються певним чином.

Якість не є абсолютною, це суб'єктивне поняття. Тому тестування, як процес своєчасного виявлення помилок та дефектів, не може повністю забезпечити коректність програмного забезпечення.

Техніка тестування також включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою умовної оцінки. Здійснюється як шляхом спостереження за його роботою в штучно створених спеціалістом ситуаціях, так і автоматизовано, на обмеженому наборі тестів, згенерованих певним чином; часто у такому режимі головною метою є перевірка виконуваності різних інваріантів, що надає змогу відстежити загальну коректність логіки програми під час виконання її різноманітних модулів, підпроцедур чи інших частин, або ж і її загалом.

Критерії оцінки тестування:

- відповідність технічному завданню, що використовувалося під час
- розробки продукту архітекторами та програмістами;
- коректність роботи на усіх можливих вхідних даних;
- коректне завершення програми за прийнятний час;
- практичність;
- сумісність із операційними системами, вказаними у технічному
- завданні;
- відповідність баченню та потребам замовника, у першу чергу таким, які були формалізовані та задокументовані у технічному завданні.

Роль тестування в розробці програмного забезпечення:

- тестування дозволяє перевірити, чи правильно реалізовано усі вимоги до ПЗ, що розроблялось.
- тестування допомагає у виявленні дефектів / помилок та забезпечує їх розпізнавання / вирішення до етапу розгортання програмного забезпечення.
- тестування пом’якшує наслідки та ризики втрат якщо програмний продукт все ж випустили по неправильних вимогах. Вимоги в такому випадку намагаються частково виправити, переоцінити. Програму покращити.
- тестування також демонструє, що створене ПЗ працює відповідає також вимогам до продуктивності.
- тестування допомагає перевірити належну інтеграцію та взаємодію програми з навколишнім середовищем.

Тестування, як і розробка програмного забезпечення також має свій життєвий цикл.

STLC (Software Testing life cycle з англ. Життєвий цикл тестування програмного забезпечення) – це дії, що необхідно виконати під час тестування програмного продукту[4]. Цей процес включає в себе такі фази:

Requirement Analysis & Clarification questions – етап аналізування вимог, необхідно скласти та затвердити документи, а також задаються уточнюючі запитання, визначаються обсяги роботи.

Test planning – цьому етапі визначається стратегія тест-плану, робиться попередня оцінка необхідних затрат часу, здійснюється вибір інструментів для тестування.

Test Design – тестовий дизайн та аналіз. На цьому етапі розробляються архітектура тест-кейсів, аналізуються та підготовуються вхідні дані для тестів і автоматизовані скрипти.

Налаштування тестового середовища – тестове середовище готується заздалегідь, воно моделює, імітує реальне довкілля, програмні оболонки.

Test execution – на цьому етапі виконуються тест-кейси, рапортується про баги та про помилки шляхом вносу у баг-трекінгові системи, відбувається повторне тестування після їх фіксення.

Test closure and reporting – відбувається підбиття підсумків, складається тест-репорт з остаточних результатів тестування, формуються тестові метрики.

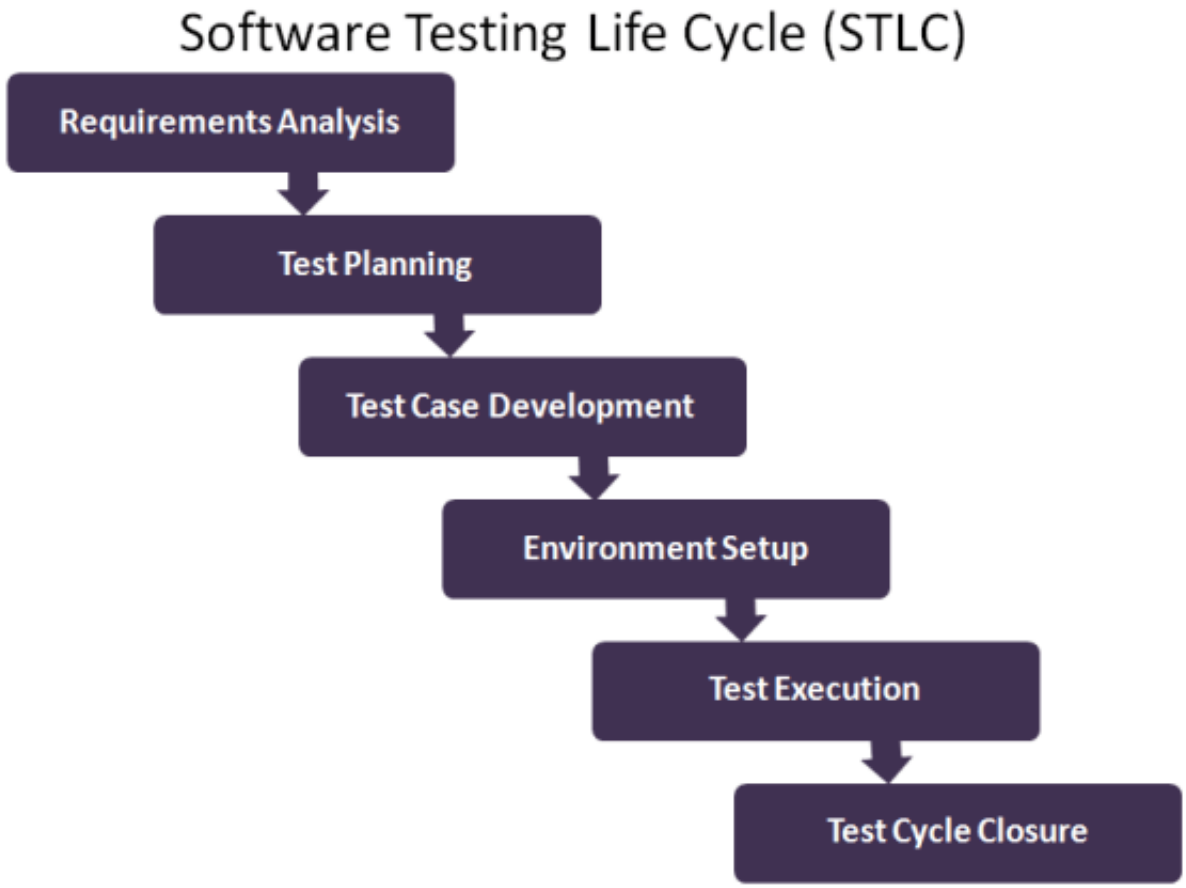

Рисунок 2.1 – Життєвий цикл тестування програмного забезпечення

Зазвичай після проходження усіх етапів проводиться аналітика та збирається статистика, яка у подальшому відправляється замовнику. Так би мовити підкреслює проведений аналіз та дозволяє обговорити нові вимоги до продукту.

Отже, тестування це досить складна система, яка постійно супроводжує розробку програмного забезпечення.

2.2 Аналіз технологій для реалізації фреймворку

Тестування якості програмного забезпечення є дуже трудомістким та відповідальним етапом при його розробці. Це обумовлює практичний інтерес до автоматизації основних процедур при тестуванні. Для автоматизованого тестування ПО використовуються різні підходи та інструменти, які здатні частково замінити участь людини при проведенні значної кількості рутинних процедур.

Для розробки фреймворку була обрана BDD технологія.

BDD використовує зрозумілі людиною описи вимог користувача до програмного забезпечення як основу для тестування програмного забезпечення. Тестери що працюють із BDD використовують спеціальний словниковий запас для створення системних тестів[5].

Кожен тест заснований на історії користувача, написаній формально визначеною мовою на основі англійської.

```
Feature: Addition
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers

  Scenario: Add two numbers
    Given I have entered 50 into the calculator
      And I have entered 70 into the calculator
     When I press add
     Then The result should be 120 on the scree
```

Рисунок 2.2 – Вигляд BDD підходу

Ця мова зосереджена виключно на діловій цінності, яку клієнт повинен отримати від програмного забезпечення, а не на описі користувальницького інтерфейсу програмного забезпечення або того, як програмне забезпечення має досягати цілей.

2.3 Аналіз інструментів для розробки

Найпопулярнішим інструментом при BDD розробці є Cucumber. Cucumber оптимізовано для BDD завдяки підтримці певного набору взаємодій між членами команди та зацікавленими сторонами.

Cucumber може виконувати функціональні специфікації (feature) як автоматизовані тести. Мова, яку розуміє Cucumber, називається Gherkin.

Cucumber підтримує написання специфікацій приблизно 30 розмовними мовами, що полегшує роботу команд за межами англомовних територій або тих, хто працює над програмним забезпеченням, орієнтованим на міжнародне значення[6].

Gherkin – мова документації, яка описує поведінку системи (BDD) за допомогою сценаріїв. Кожен сценарій написаний зрозумілою для бізнесу (недосвідченого користувача) мовою та повністю описує якусь частину функціоналу[7].

Для розробки фреймворку буде використано Cucumber, а працювати із ним будемо за допомогою AppiumGUI.

Appium – це безкоштовний кросплатформовий інструмент з відкритим вихідним кодом, який допомагає автоматизувати програми як для Android, так і для iOS. Appium дотримується того ж підходу, що і Selenium WebDriver, який отримує HTTP-запити у форматі JSON від клієнтів та перетворює їх залежно від платформи, на якій він працює[8].

Для того щоб працювати із Appium та створювати тести на Android пристрої необхідно:

- Java (version > 7)
- Android SDK (version > 17)
- Android virtual/real device

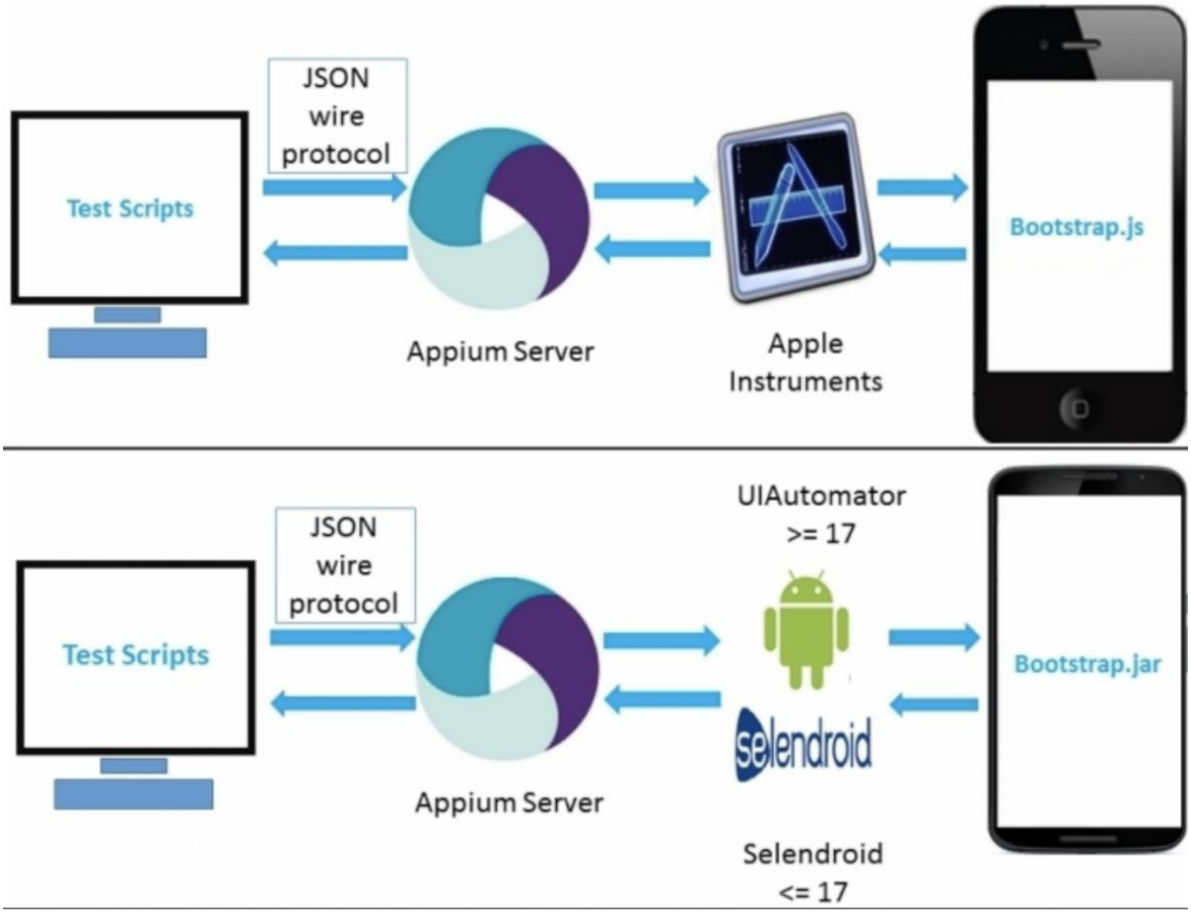

Рисунок 2.3 – Схема роботи Appium

Для того щоб створювати тести для iOS необхідно:

- MacOS X (version >= 10.7)
- Xcode (version >= 5.1)
- Java (version > 7)
- Homebrew
- Npm
- iOS device

Тому через технічні можливості тестування мобільного додатку буде проводитись на базі Android систем.

Для Android наразі існує безліч інструментів для автоматизації тестування, багато з них є кроссплатформенними. Серед них (тестування клієнтської частини): Appium, Robotium, UI Automator, XCUITest, Eggplant, Ranorex та інші. Postman, RestAssured, SoapUI, Fiddler – для автоматизації тестування серверної частини.

Існують й інші, більш вузькоспеціалізовані утиліти для автоматизації тестування певних частин проектів. Наразі на ринку мобільних застосувань продовжують з’являтися усе нові технології та усе нові пристрої, що ускладнює процес тестування та збільшує кількість необхідних тестів.

2.4 Висновки

1. Визначено критерії оцінки якості тестування, показано, що тестування є невід’ємною частиною розробки програмного забезпечення і автоматизація тестування є вкрай важливою.
2. Аналіз інструментів реалізації BDD-методології показав, що найбільші переваги має спеціалізований фреймворк Cucumber, мова документації - Gherkin, яка описує поведінку системи (BDD) за допомогою сценарії, а також Appium-сервер для реалізації з’єднання з мобільним пристроєм.

3 РОЗРОБКА ФРЕЙМВОРКУ ДЛЯ ТЕСТУВАННЯ МОБІЛЬНОГО ДОДАТКУ JETIQ

3.1 Варіантний аналіз та обґрунтування вибору засобів для реалізації програмного продукту

Перед початком розробки необхідно обрати інструменти та засоби розробки програмного забезпечення.

Для розробки фреймворку була обрана мова програмування Java.

Java – об'єктно-орієнтована мова програмування, випущена 1995 року компанією «Sun Microsystems» як основний компонент платформи Java. З 2009 року мовою займається компанія «Oracle», яка того року придбала «Sun Microsystems». В офіційній реалізації Java-програми компілюються у байт-код, який при виконанні інтерпретується віртуальною машиною для конкретної платформи. Мова значно запозичила синтаксис із C і C++. Зокрема, взято за основу об'єктну модель C++, проте її модифіковано. Усунуто можливість появи деяких конфліктних ситуацій, що могли виникнути через помилки програміста та полегшено сам процес розроблення об'єктно-орієнтованих програм. Передусім Java розроблялась як платформо-незалежна мова, тому вона має менше низькорівневих можливостей для роботи з апаратним забезпеченням, що в порівнянні, наприклад, з C++ зменшує швидкість роботи програм. За необхідності таких дій Java дозволяє викликати підпрограми, написані іншими мовами програмування[9].

Програмний код можна писати у різноманітних середовищах розробки, тому необхідно провести аналіз середовищ розробки на мові Java. Для порівняльного аналізу було обрано Eclipse, NetBeans, IntelliJIDEA.

Eclipse – вільне модульне інтегроване середовище розробки програмного забезпечення (рисунок 3.1). Розробляється і підтримується Eclipse Foundation і включає проєкти, такі як платформа Eclipse, набір інструментів для програмістів на мові Java, системи контролю версій, конструктори GUI тощо. Написаний в основному на Java, може бути використаний для розробки

застосунків на Java і, за допомогою різних плагінів, на інших мовах програмування, включаючи Ada, C, C++, C#, COBOL, Fortran, Perl, PHP, Python, R, Ruby (включно з каркасом Ruby on Rails), Scala, Clojure та Scheme. Середовища розробки зокрема включають Eclipse ADT (Ada Development Toolkit) для Ada, Eclipse CDT для C/C++, Eclipse JDT для Java, Eclipse PDT для PHP[10].

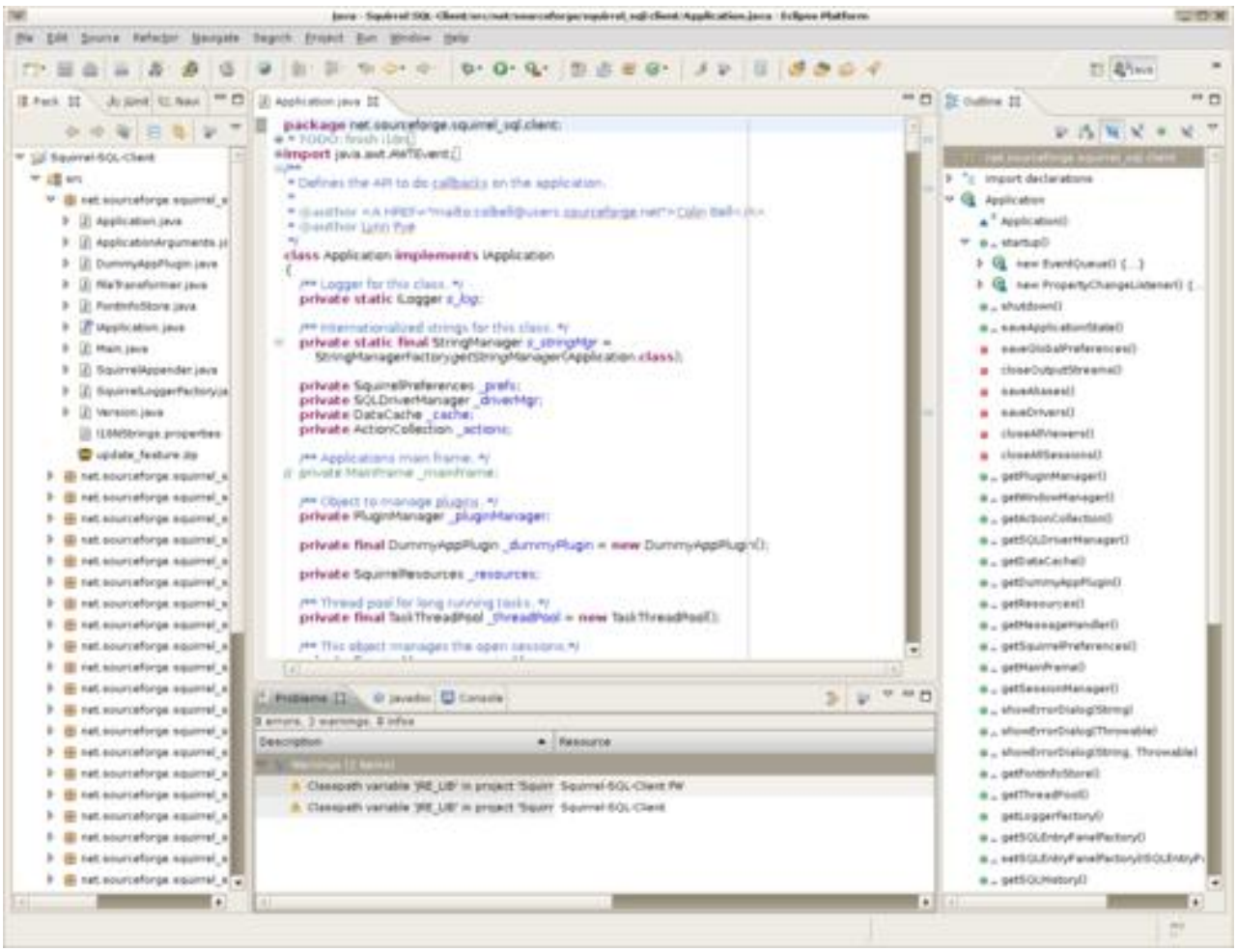

Рисунок 3.1 – Інтерфейс середовища розробки Eclipse

NetBeans (рисунок 3.2) – вільне інтегроване середовище розробки (IDE) для мов програмування Java, JavaFX, C/C++, PHP, JavaScript, HTML5, Python, Groovy. Середовище може бути встановлене і для підтримки окремих мов, і у повній конфігурації. Проєкт NetBeans IDE підтримувався і спонсорувався фірмою Sun Microsystems і після придбання Sun – Oracle. У жовтні 2016 року

Oracle передав NetBeans у власність Apache Software Foundation, яка займається розробкою і підтримкою проєкту. Останні версії NetBeans IDE підтримують рефакторинг, профілювання, виділення синтаксичних конструкцій кольором, автодоповнення конструкцій, що набираються на льоту, і безліч визначених шаблонів коду. NetBeans IDE доступна у вигляді готових дистрибутивів (прекомпільованих бінарних файлів) для платформ Microsoft Windows, Linux, FreeBSD, Mac OS X, OpenSolaris та Solaris (як для SPARC, так і для x86 – Intel та AMD). Для решти платформ доступна можливість скомпілювати NetBeans самостійно з вихідних текстів. NetBeans IDE підтримує плагіни, дозволяючи розробникам розширювати можливості середовища[11].

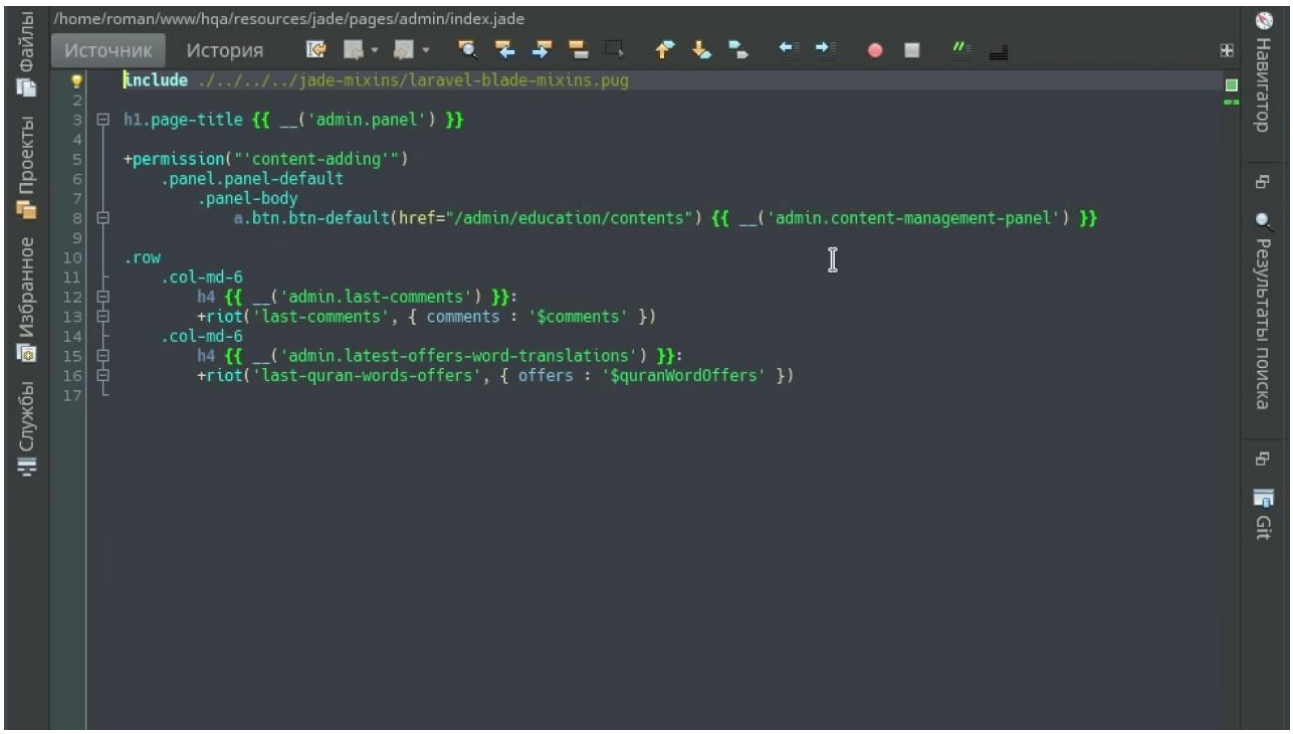

Рисунок 3.2 – Інтерфейс середовища розробки NetBeans

IntelliJ IDEA – комерційне інтегроване середовище розробки для різних мов програмування від компанії JetBrains (рисунок 3.3). Система поставляється у вигляді урізаної по функціональності безкоштовної версії «Community Edition» і повнофункціональної комерційної версії «Ultimate Edition», для якої активні розробники відкритих проектів мають можливість отримати

безкоштовну ліцензію. Перша версія IntelliJ IDEA з'явилася у січні 2001 року й швидко здобула популярність, як перша Java IDE із широким набором інтегрованих інструментів для рефакторингу що дозволяла програмістам швидко реорганізовувати початковий код програм. Дизайн середовища орієнтовано на продуктивність праці програмістів, дозволяючи їм сконцентруватися на розробці функціональності, тоді як IntelliJ IDEA бере на себе виконання рутинних операцій.

Community версія середовища IntelliJ IDEA підтримує інструменти (у вигляді плагінів) для проведення тестування TestNG і JUnit, системи контролю версій CVS, Subversion, Mercurial і Git, засоби складання Maven, Ant, Gradle, мови програмування Java, Scala, Clojure, Groovy і Dart. Підтримується розробка застосунків для мобільної платформи Android. До складу входить модуль візуального проектування GUI-інтерфейсу Swing UI Designer, XML-редактор, редактор регулярних виразів, система перевірки коректності коду, система контролю за виконанням завдань і доповнення для імпорту та експорту проектів з Eclipse.

Комерційна версія «Ultimate Edition» відрізняється наявністю підтримки додаткових мов програмування (наприклад, PHP, Ruby, Python, JavaScript, CoffeeScript, HTML, CSS, SQL), підтримкою технологій Java EE, UML-діаграм, підрахунок покриття коду[12].

Порівнявши середовища розробки, через найбільшу зручність та можливість встановлення плагінів – що необхідні при розробці фреймворку для тестування. У якості середовища розробки було обрано IntelliJ IDEA.

Для тестування мобільного додатку також необхідно мати реальний мобільний пристрій або спосіб його замінити.

Для вирішення цієї задачі обрано Android Studio так як середовище має вбудовану функцію емуляції.

Емуляція дозволяє виконувати комп'ютерну програму на платформі, відмінній від тієї, для якої вона була написана в оригіналі. Емуляцією також називають сам процес цього виконання (рисунок 3.4).

Android Studio – інтегроване середовище розробки (IDE) для платформи Android, представлене 16 травня 2013 року на конференції Google I/O менеджером по продукції корпорації Google – Еллі Паверс. Середовище побудоване на базі вихідного коду продукту IntelliJ IDEA Community Edition, що розвивається компанією JetBrains. Android Studio розвивається в рамках відкритої моделі розробки та поширюється під ліцензією Apache 2.0 (рисунок 3.5).

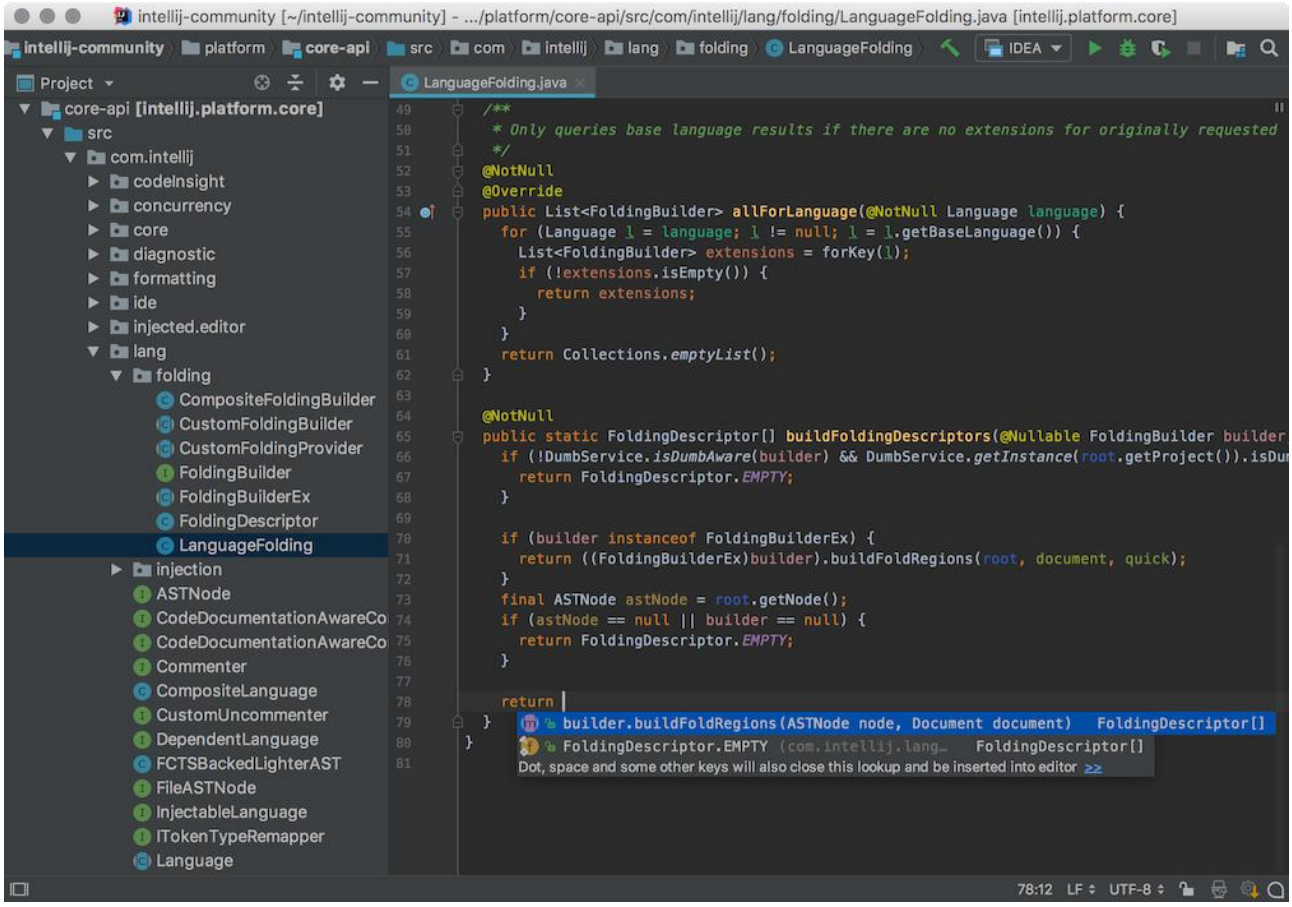

Рисунок 3.3 – Інтерфейс середовища розробки IntelliJ IDEA

Середовище розробки адаптоване для виконання типових завдань, що вирішуються в процесі розробки застосунків для платформи Android.

У тому числі у середовище включені засоби для спрощення тестування програм на сумісність з різними версіями платформи та інструменти для проектування застосунків, що працюють на пристроях з екранами різної роздільності (планшети, смартфони, ноутбуки, годинники, окуляри тощо).

Крім можливостей, присутніх в IntelliJ IDEA, в Android Studio реалізовано кілька додаткових функцій, таких як нова уніфікована підсистема складання, тестування і розгортання застосунків, заснована на складальному інструментарії Gradle і підтримуюча використання засобів безперервної інтеграції[13].

Рисунок 3.4 – Вікно емуляції Android пристрою

Отже, проаналізувавши необхідні інструменти та засоби розробки фреймворку для реалізації було обрано середовище IntelliJ IDEA та Android Studio для емуляції мобільного пристрою, мову програмування Java.

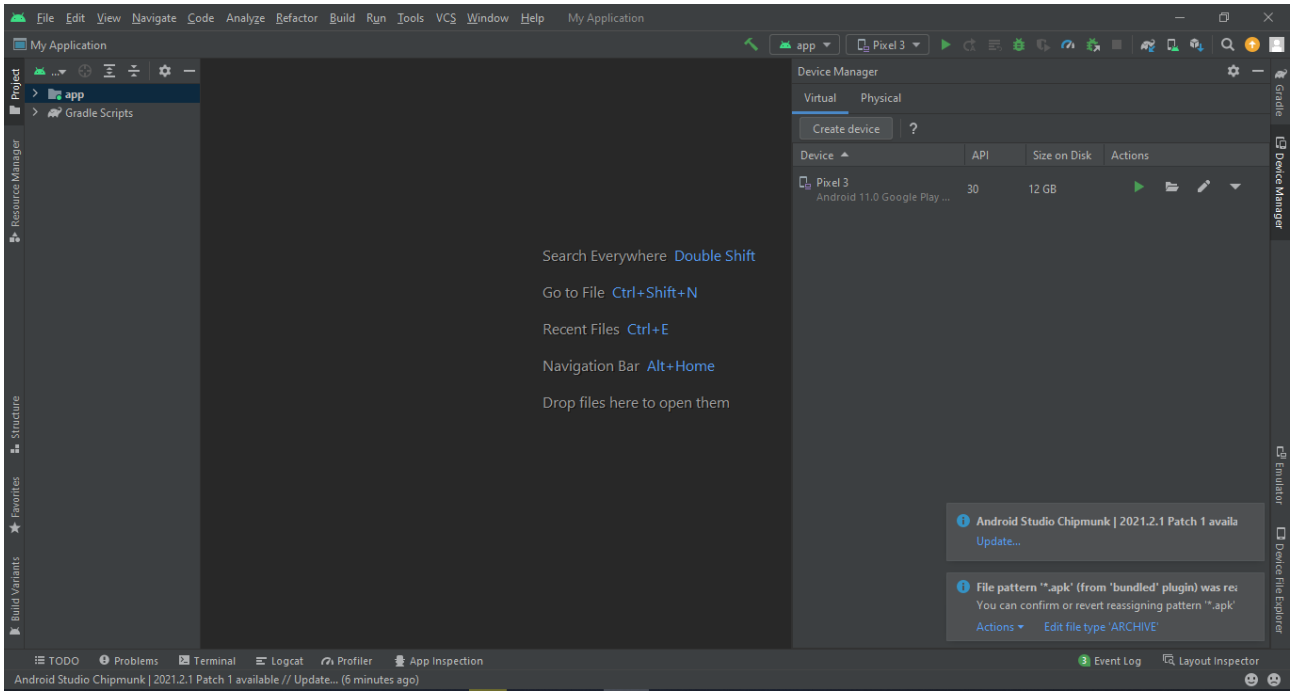

Рисунок 3.5 – Інтерфейс середовища розробки Android Studio

3.2 Розробка методу та програмного коду автоматизованого тестування

Згідно до обраного підходу розробки BDD створення тестів починається з опису поведінки, яку сценарій повинен перевірити. Після чого створюємо методи що дозволять покрити цю поведінку. Перш за все необхідно створити об’єкти що є на сторінках додатка, які будуть використовувати та вказати локатори (так звані шляхи до об’єкту що дозволять програмному коду взаємодіями з об’єктом) до них. Наступним кроком буде саме розробка методів для покриття необхідного функціоналу та використання методів у кроках, описаних у feature файлах. Для розробленого фреймворку алгоритм може бути доволі простим, хоча на великих проектах в алгоритмі також приймають участь аналіз змін що вносять розробники та систематична перевірка на різноманітного роду проблеми. А також подальша підтримка створених тестів командою тестувальників.

Приклад алгоритму для проекту (рисунок 3.6):

1. Описати поведінку – це включає в себе потік і особливості продукту означає основне бачення.

2. Визначте вимоги – змодельовані вимоги з бізнес-правилами для спільного розуміння.
3. Виконати і провалити тести – розробляйте та запускайте тестові випадки.
4. Застосувати оновлення коду – реорганізуйте його відповідно до вимог.
5. Виконати тести із вдалим результатом – запустіть оновлений код і пройдіть тестові випадки.

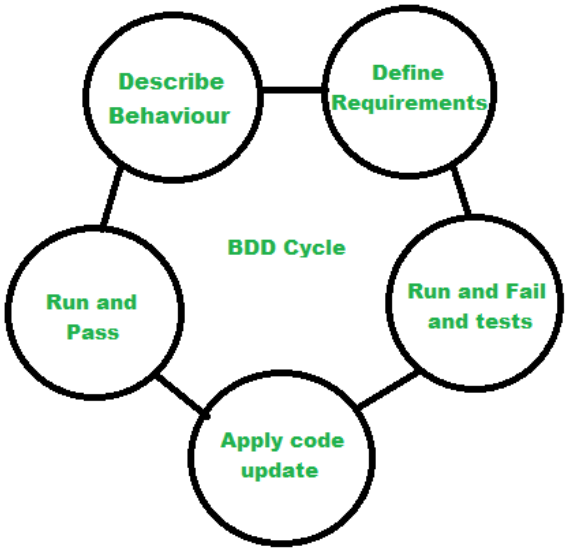

Рисунок 3.6 – Приклад BDD алгоритму

Оцінивши дії що необхідні для створення тестів у фреймворкі у результаті маємо такі кроки алгоритму:

1. Описати поведінку у feature файлі;
2. Підготувати локатори для необхідних елементів сторінки;
3. Розробити методи для взаємодії із елементами сторінки;
4. Описати кроки сценарію з feature файлу за допомогою розроблених методів;
5. Запустити тест.

Блок-схема алгоритму зображена на рисунку 3.7.

Рисунок 3.7 – Блок-схема алгоритму створення автоматизованих тестів

Отже, було представлено алгоритм створення автоматизованих тестів.

Загальна схема тестування наведена на рисунку 3.8, а на рисунку 3.9 представлено діаграму класів фреймворку для тестування мобільного додатку JetIQ.

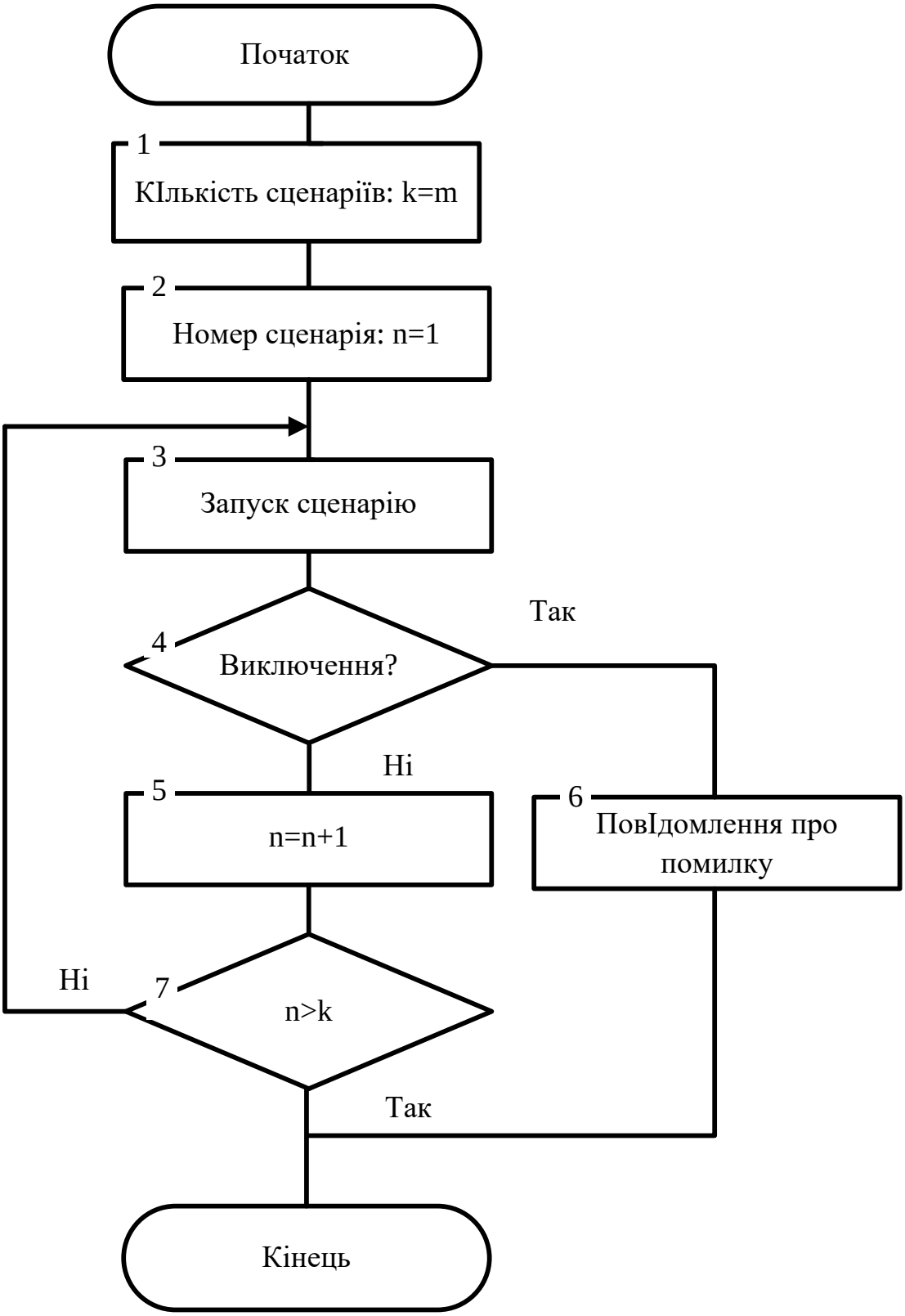

Рисунок 3.8 – Блок-схема алгоритму автоматизованого тестування

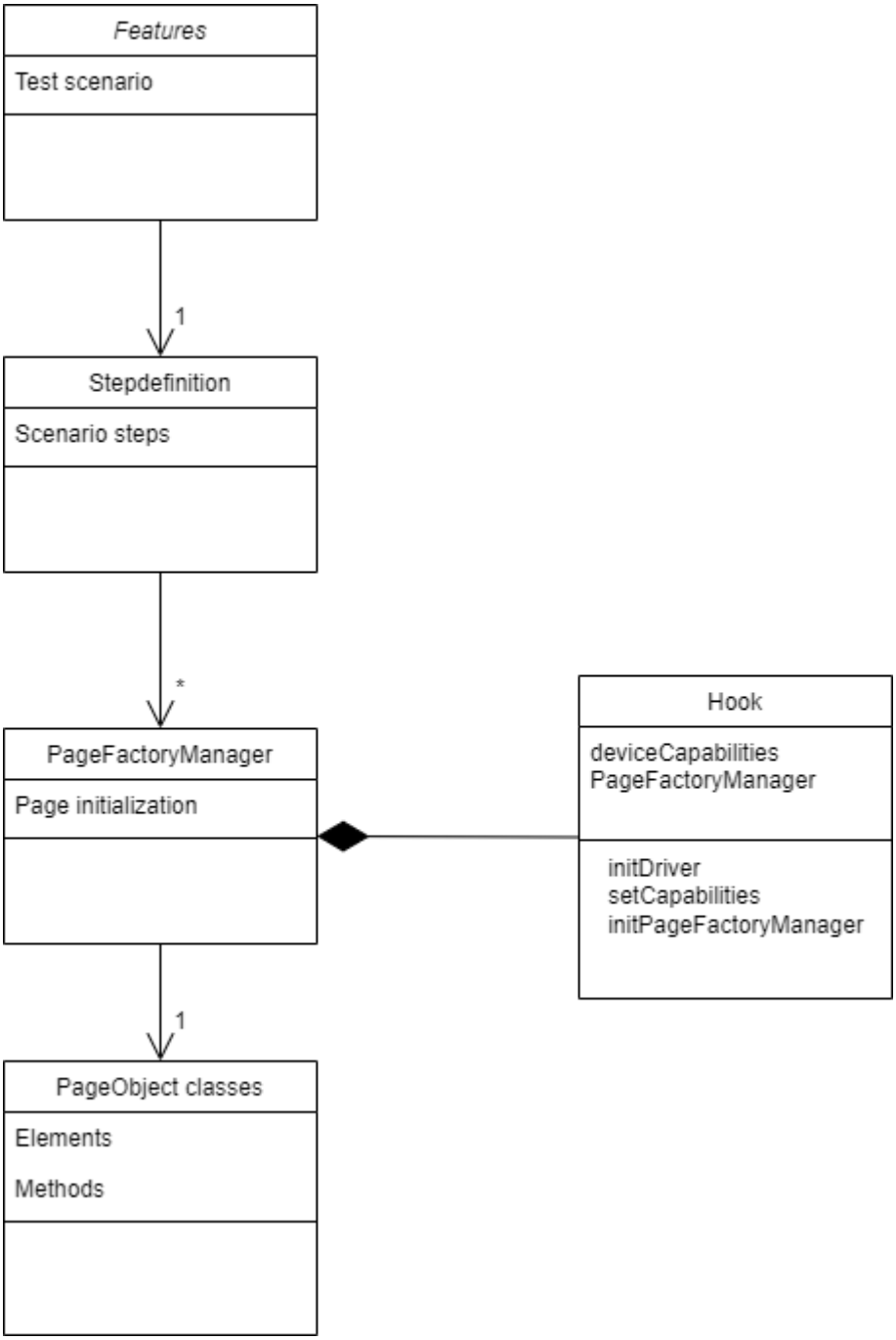

Рисунок 3.9 – Діаграма класів

На діаграмі класів представлені залежності та взаємодії між класами фреймворку. Першочергово у feature файлах розроблені сценарії, що складаються із кроків. Кроки звертаються до stepdefinitions класів, у яких описана логіка їх роботи.

У stepdefinitions файлах виклакається PageFactoryManager. Його необхідно ініціалізувати із класу Hooks. У Hooks класі записуються

налаштування девайсу (capabilities), ініціалізується драйвер у який ці налаштування записуються. Драйвер передається у екземпляр PageFactoryManager при його ініціалізації. Після чого із виклику PageFactoryManager програмний код розуміє до якого класу PageObject необхідно звернутись, які елементи і методи використати.

3.3 Висновки

1. Для розробки фреймворку тестування мобільного додатку JetIQ обрано середовище розробки IntelliJ IDEA, середовище для проведення емуляції пристрою на базі операційної системи Android – Android Studio та мову програмування Java, оскільки ці інструменти в повній мірі відповідають принципам RAD.
2. Вперше запропоновано метод автоматизації тестування мобільного додатку JetIQ, особливість якого полягає у використанні фреймворку, що дозволяє підвищити швидкість тестування мобільного додатку JetIQ в 5-10 разів у порівнянні з ручним тестуванням.
3. Подальшого розвитку отримав метод розробки фреймворку для тестування мобільного додатку JetIQ, у якому, на відміну від існуючих, використано BDD-методологію розробки програмного забезпечення, що спростило розробку тестів за рахунок побудови сценаріїв поведінки.
4. Мовою програмування Java розроблено фреймворк для тестування мобільного додатку JetIQ та сценарії тестів з використанням мови документації Gherkin та фреймворку Cucumber.

4 ТЕСТУВАННЯ МОБІЛЬНОГО ДОДАТКУ JETIQ ЗА ДОПОМОГОЮ РОЗРОБЛЕНОГО ФРЕЙМВОРКУ

4.1 Аналіз розробленого фреймворку

У ході розробки були створені сценарії для тестування у так званих feature файлах. Для опису сценаріїв використовується мова документації Gherkin, яка містить ряд ключових слів для побудови сценаріїв, що взаємодіють з фреймворком Cucumber. Розглянемо основні ключові слова мови Gherkin на прикладі побудови сценарію тестування логіну.

Опис сценарію починається ключовим словом «As a user», після якого з нового рядку описується побажання користувача щодо функціоналу тестування.

Наступним по порядку виконується оголошення сценарію з використанням ключового слова «Scenario» (у випадку не використання табличних значень) або «Scenario Outline» (дозволяє використовувати табличні значення), наприклад:

Scenario Outline: Verify log-in workflow with login

Далі описується власне тест, який завжди починається словом «Given». Для опису тесту можуть використовуватись такі ключові слова:

- When - подія, яка ініціює цей сценарій. Якщо подію не можна розкрити однією пропозицією, всі наступні деталі розкриваються через ключові слова And і But;
- Then - результат, який користувач повинен спостерігати, якщо результат не можна розкрити однією пропозицією, всі наступні деталі розкриваються через ключові слова And і But;
- And - допоміжне ключове слово, аналог кон'юнкції;
- But - допоміжне ключове слово, аналог заперечення.

Сценарій записується в текстовий файл з розширенням .feature, приклад вмісту цього файлу наведено на рисунку 4.1.

```
As a user
I want to be able to log in into my account
So that i can be sure that functional log in form works correctly

Scenario Outline: Verify log-in workflow with login
  Given  User enter login into login field
  When User enter '<correctPassword>' into password field
  And User clicks on 'sign in' button
  Then User sees 'Auto-filling register' alert
  When User doesn't give permission for auto filling register
  Then User sees personal account
  Examples:
    | correctPassword |
    | 9293            |
```

Рисунок 4.1 – Приклад тестових сценаріїв

Таблична змінна correctPassword містить пароль для кроку введення паролю і її значення описується під ключовим словом «Examples» як показано на рисунку 4.1

Для кроків що були використані у сценаріях розроблено stepdefinitions класи, у яких прописується логіка кроків для розроблених тестів:

- клас LoginStepDefs – виконує сценарії тестування функціоналу логіну;
- клас SmokeStepDefs – тестування основних функцій додатку.

Клас LoginStepDefs містить такі методи:

- userClicksOnBackButton() – реалізує повернення на головну сторінку;
- void userPerformLogin() – виконує вхід до персонального кабінету студента JetIq;
- userEnterNicknameIntoNicknameField () – введення нікнейм при входженні в профіль в JetIq;
- userEnterLoginIntoLoginField() – введення ідентифікатора студента;

- userEnterPasswordIntoPasswordField(String password) – введення паролю студента;
- userClicksOnSignInButton() – натискання на кнопку входу в профіль;
- userSeeAutofillRegisterAlert() – перевірка відображення повідомлення про авто заповнення журналу;
- userGivePermissionForAutoFillingRegister() – надання дозволу на автозаповнення журналу;
- userDoesNotGivePermissionForAutoFillingRegister() - заборона дозволу на автозаповнення журналу;
- userSeesPersonalAccount() – перевірка відображення профілю студента.

Функції класу SmokeStepDefs реалізують 36 методів. Деякі з них такі:

- userOpensNavigationMenu() – відкриває меню навігації мобільного додатку JetIQ;
- userSeesNavigationMenu() – перевірка відображення меню навігації;
- userSeesProfileNavigationButton() – перевірка відображення кнопки профілю у меню навігації;
- userSeesNotificationsNavigationButton() - перевірка відображення кнопки повідомлень у меню навігації;
- userSeesScheduleNavigationButton() - перевірка відображення кнопки розкладу у меню навігації;
- userOpensNotificationsPage() – відкриття сторінки в мобільному додатку JetIQ з повідомленнями;
- userOpensSchedulePage() - відкриття сторінки розкладу в мобільному додатку JetIQ;
- userCheckFacultyInformationContainsTextFaculty() – перевірка інформації про факультет студента;

- userCheckGroupInformationContainsTextGroup() – перевірка інформації про групу студента;
- userCheckCourseInformationContainsTextCourse() - перевірка інформації про рік навчання студента;

Клас PageFactoryManager, який реалізацією singleton паттерну - менеджер за допомогою якого виконується звернення до розроблених класів сторінок Page Object, а також записується та зберігається налаштування драйвера. Методи цього класу такі:

- getDriver() – метод отримання драйвера;
- getInstance() – створення єдиного екземпляру драйвера;
- setDriver() – встановлення драйвера;
- getLogInPage() – виклик сторінки логіну;
- getHomePage() – виклик головної сторінки мобільного додатку JetIQ;
- getNotificationsPage() - виклик сторінки повідомлень мобільного додатку JetIQ;
- getSchedulePage() - виклик сторінки розкладу мобільного додатку JetIQ;
- getOldMessengerPage() - виклик сторінки месенджеру мобільного додатку JetIQ;
- getMaterialsPage() - виклик сторінки навчальних матеріалів студента.

Класи сторінок мобільного додатку PageObject класи містять локатори елементів сторінок та методи для взаємодії із ними.

Клас BasePage містить базові методи, що є актуальними для усіх сторінок. BasePage наслідують усі PageObject класи.

Клас Hooks містить попереднє налаштування та ініціалізацію драйвера.

Лістинг коду методів цих класів наведено в додатку В.

4.2 Тестування

Виконавши запуск деякої кількості демонстраційних тестів було отримано наступні результати. Результати проведеного тестування зображені на рисунку 4.2 та 4.3.

Рисунок 4.2 – Результат виконання тестів

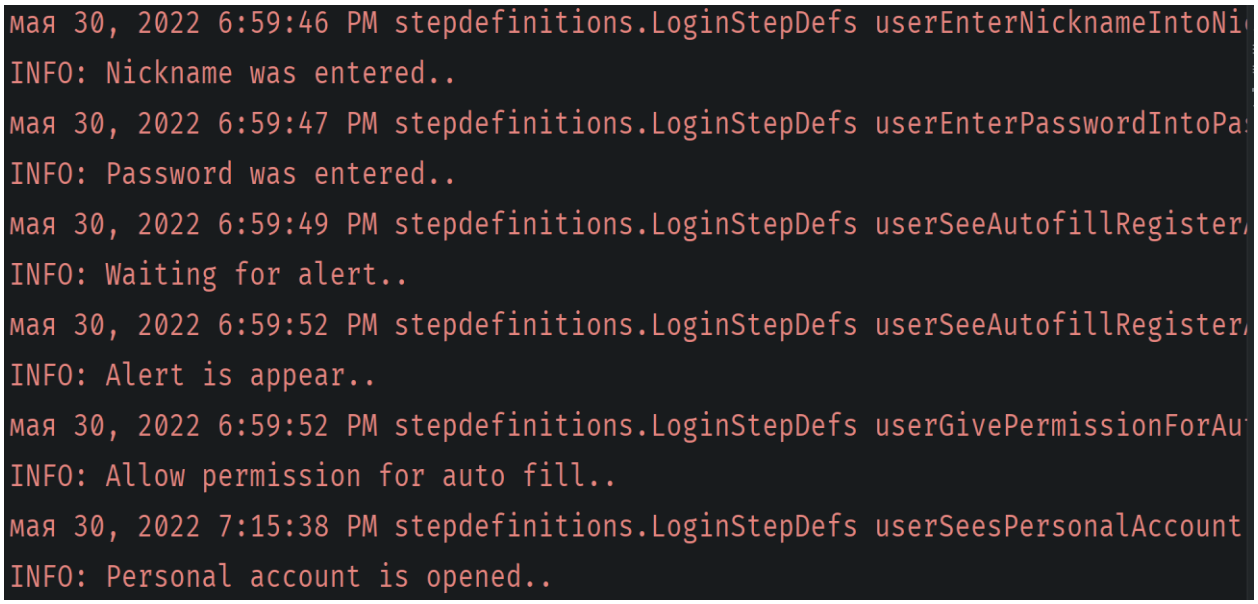

Рисунок 4.3 – Логи виконаного тесту

Як бачимо із результатів тести пройшли успішно. Процес автоматизованого тестування представлено на рисунках 4.4- 4.6.

Рисунок 4.4 – Процес тестування, заповнення полів даними

На рисунку 4.5 представлено момент заповнення поля логіну із особистими даними користувача.

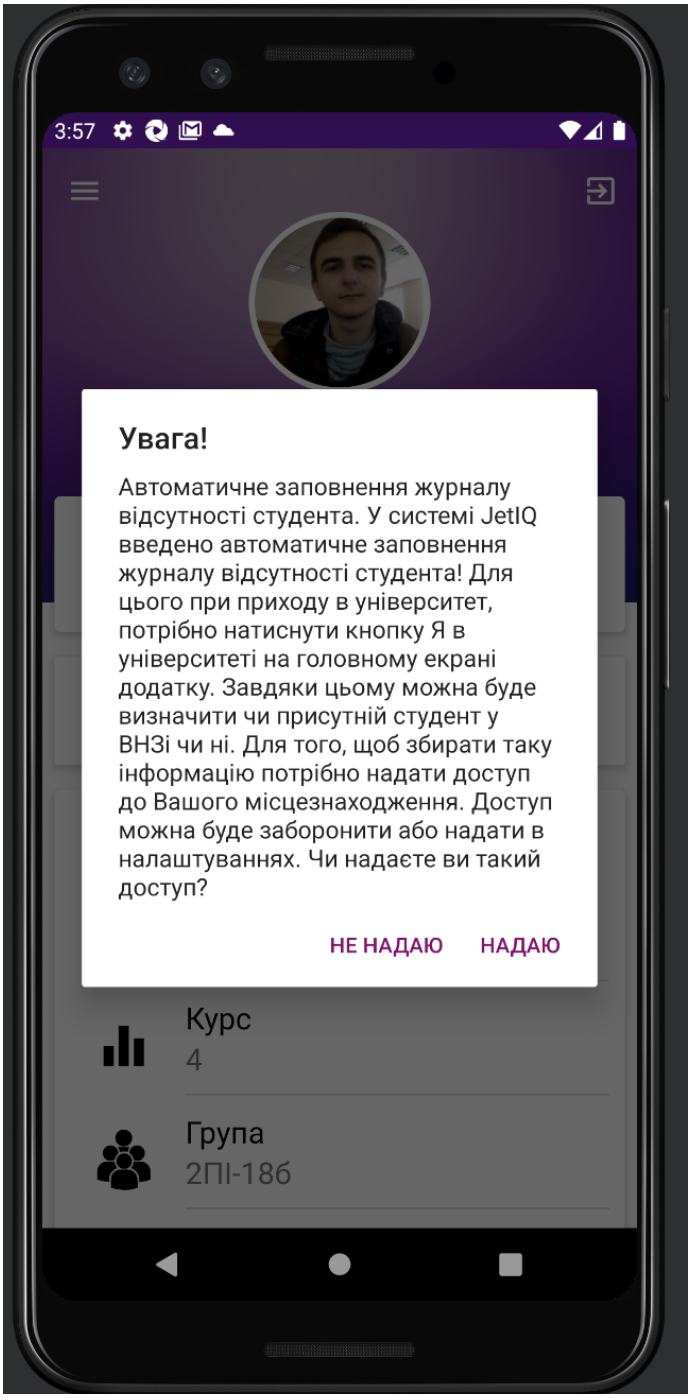

Рисунок 4.5 – Процес тестування, перевірка повідомлення

На рисунку 4.5 зображено перевірку відображення повідомлення про автоматичне заповнення журналу. На рисунку 4.6 зображено головний екран, що є підтвердженням факту завершення тестування. Висновком якого є те що функціонал логіну у додатку функціонує як слід.

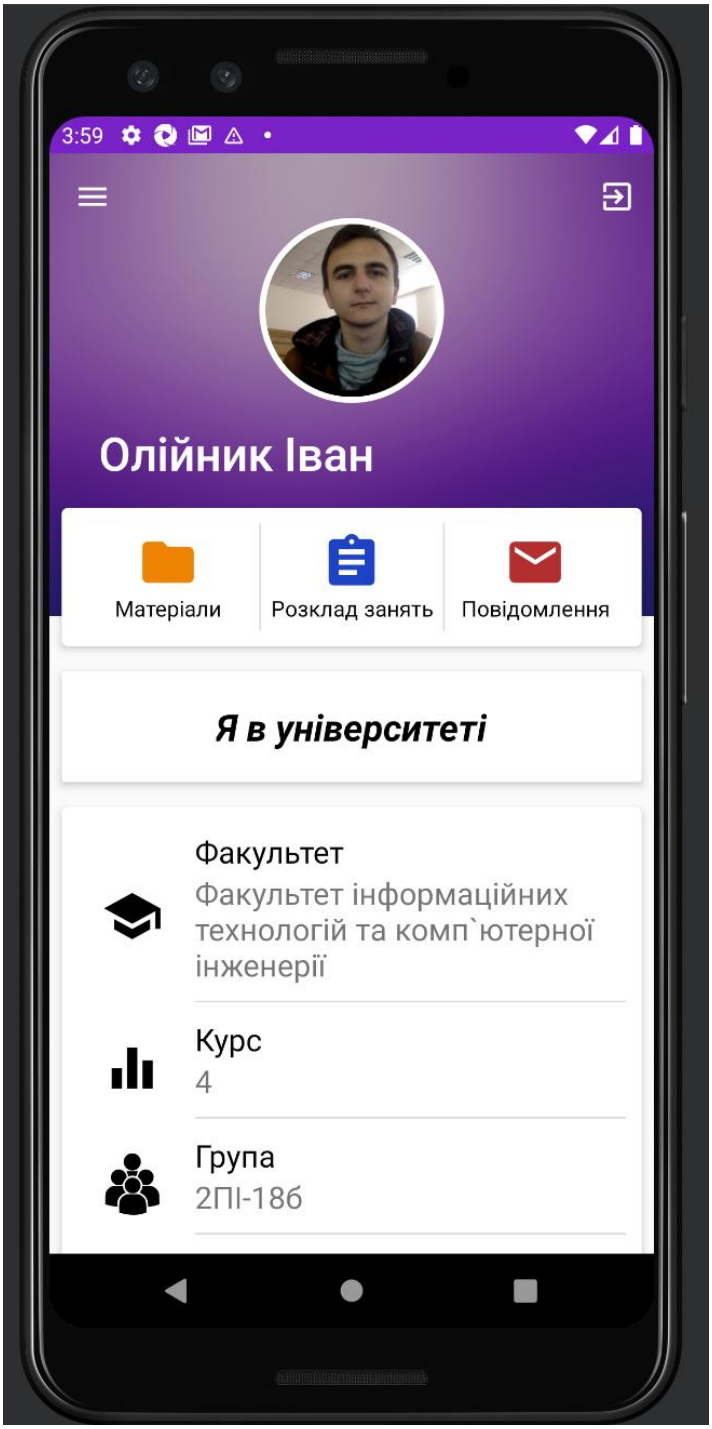

Рисунок 4.6 – Головний екран додатку

Приведемо для прикладу ще один тест, який перевіряє наповнення сторінки профілю та функціювання кнопок що знаходяться на ній. Початок тесту для уникнення дублікатів зображено на рисунку 4.6.

На цій сторінці перевіряється наявність інформації про факультет, групу та курс. А також наявність кнопок «Матеріали», «Розклад занять» та «Повідомлення».

Наступним кроком натискається кнопка відкриття меню навігації додатку та перевіряємо що він відкрився. Перевірка зображена на рисунку 4.7.

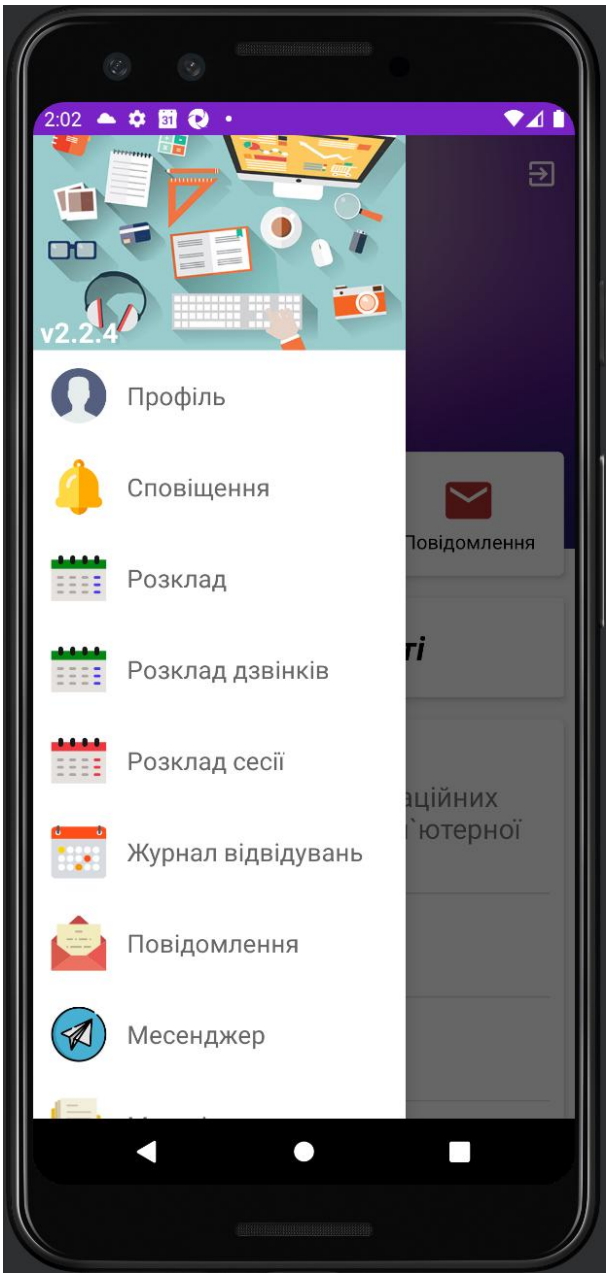

Рисунок 4.7 – Перевірка відображення меню навігації

Після чого меню навігації закривається. Натискається кнопка навчальних матеріалів та перевіряється чи відкрита сторінка із матеріалами. Перевірку сторінки матеріалів зображено на рисунку 4.8.

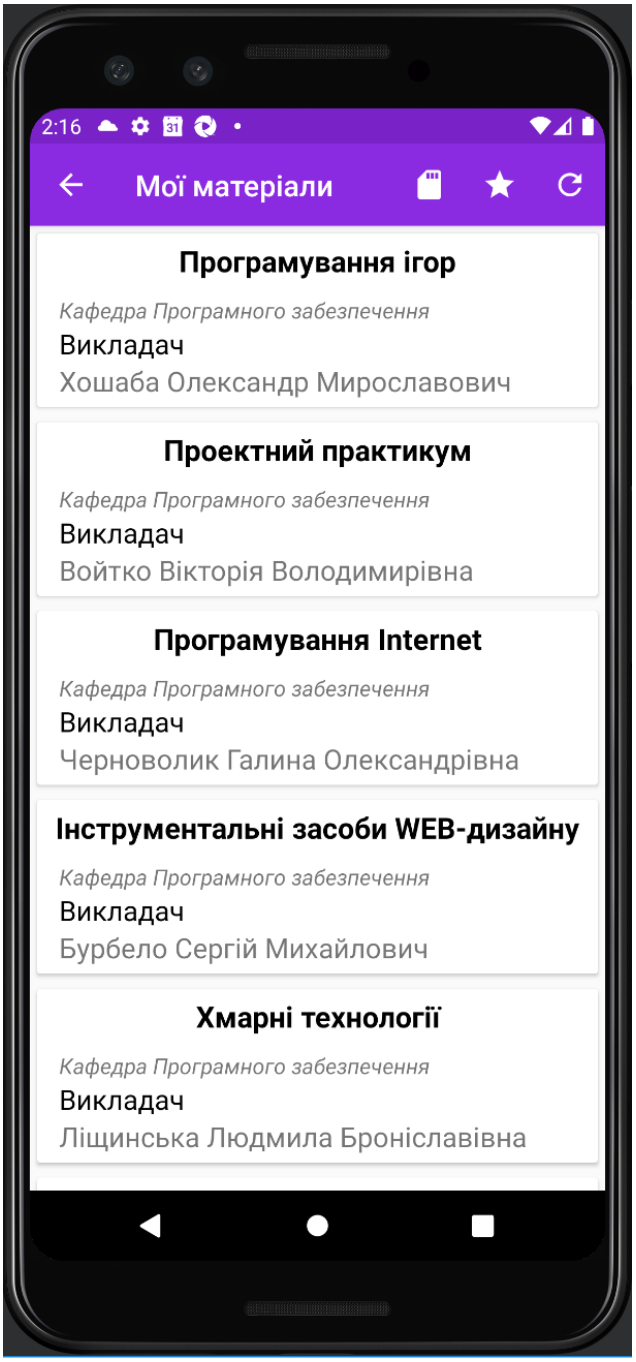

Рисунок 4.8 – Перевірка відображення сторінки матеріалів

Наступним кроком повертаємось на сторінку профіля натиснувши на кнопку «Назад», у лівому верхньому куті та натискаємо на кнопку «Розклад занять», після чого перевіряємо сторінку розкладу. Перевірку сторінки розкладу зображено на рисунку 4.9.

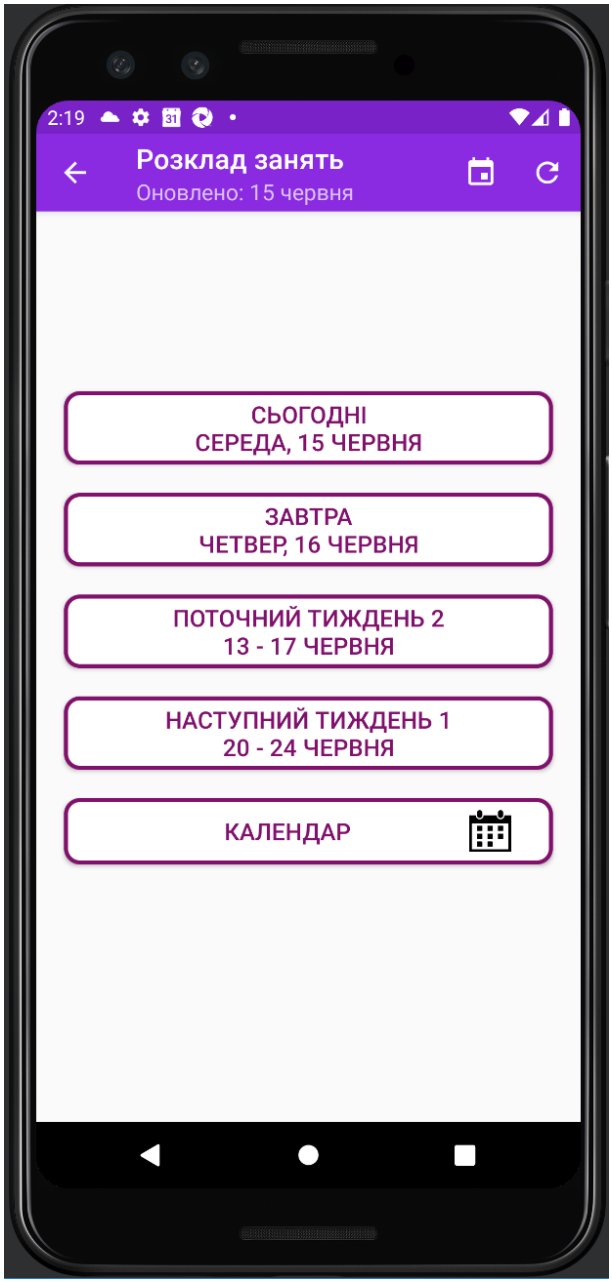

Рисунок 4.9 – Перевірка відображення сторінки розкладу

Наступним кроком тесту повертаємось на сторінку профіля, натискаємо на кнопку «Повідомлення» та перевіряємо що сторінка відобразилася. Перевірку відображення сторінки повідомлень зображено на рисунку 4.10.

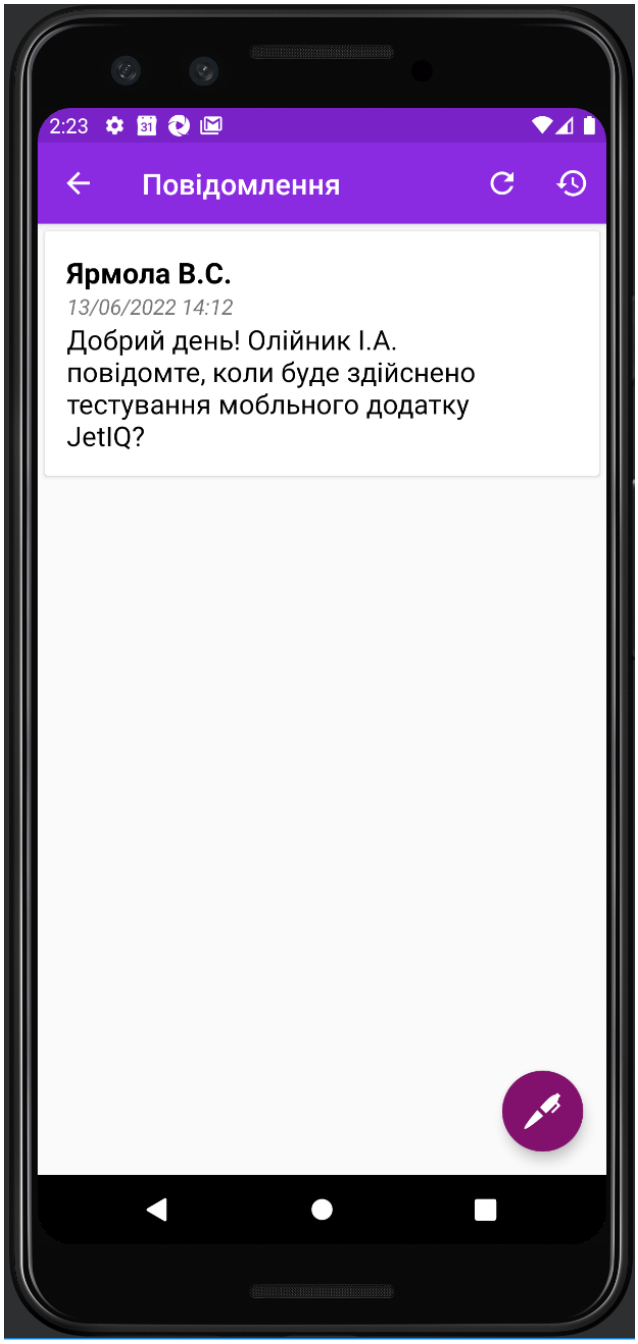

Рисунок 4.10 – Перевірка відображення сторінки повідомлень

Також перевірено роботу основних функцій додатку: працездатність та наповнення сторінок мобільного додатку та коректна робота меню додатку.

4.3 Інструкція користувача

Інструкція користувача розроблена та представлена для налаштування та використання фреймворку подальшими розробниками. Першочергово використовувалась мова програмування JAVA, тому необхідне середовище

розробки повинно підтримувати цю мову. У даній інструкції для використання представлено середовище IntelliJ IDEA.

Для використання фреймворку необхідний наступний набір інструментів.

Для того щоб створювати тести для Android пристроїв необхідно:

- Java (version > 7);
- Android SDK (version > 17);
- Appium;
- Android virtual/real device.

Розглянемо кроки для налаштування засобів тестування:

- Встановити середовище розробки;
- Завантажити та встановити JDK;
- Встановити Anroid Studio;
- Завантажити та встановити Anroid SDK;
- Налаштувати всередині Andriod Studio, емулятор мобільного пристрою;
- Запустити емулятор;
- Встановити на емулятор мобільного пристрою мобільний додаток JetIQ;
- Встановити та налаштувати Appium. У нових версіях Appium, для пошуку локаторів елементів додатку, що тестується, необхідно окремо встановити Appium Inspector (налаштування Appium та інспектора зображено на рисунках 4.11 та 4.12);
    - Встановити Appium Inspector
- Запустити фреймворк;
- Запустити тести за допомогою TestRunner для того щоб переконатись у працездатності програмного коду.

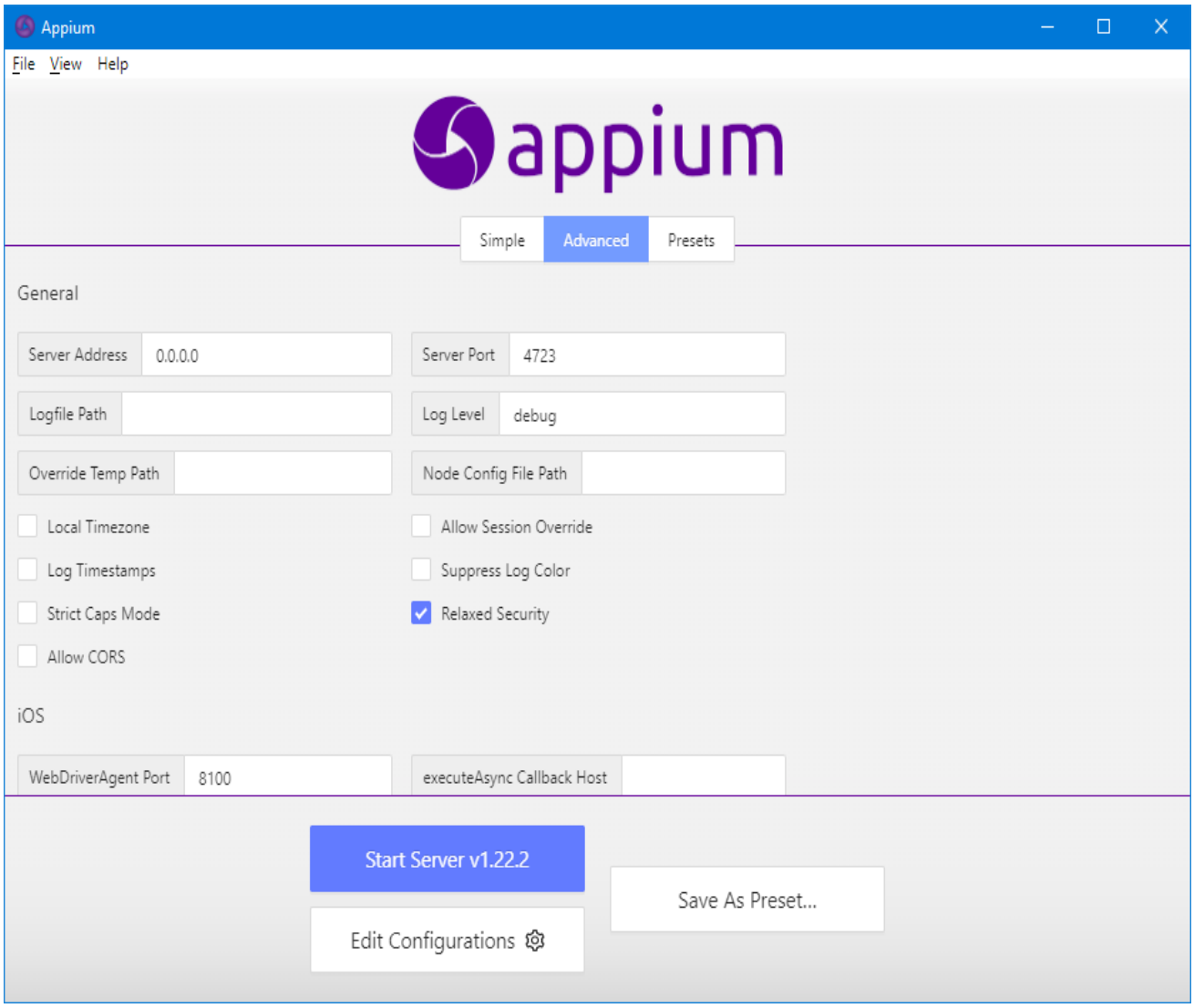

Рисунок 4.11 – Вікно налаштування Appium

На рисунку 4.11 зображено використані налаштування зокрема: адреса серверу 0.0.0.0 та порт 4723, що автоматично налаштовані та рідно потребують зміни та можливість модифікувати з’єднання за допомогою перемикання чек-боксів.

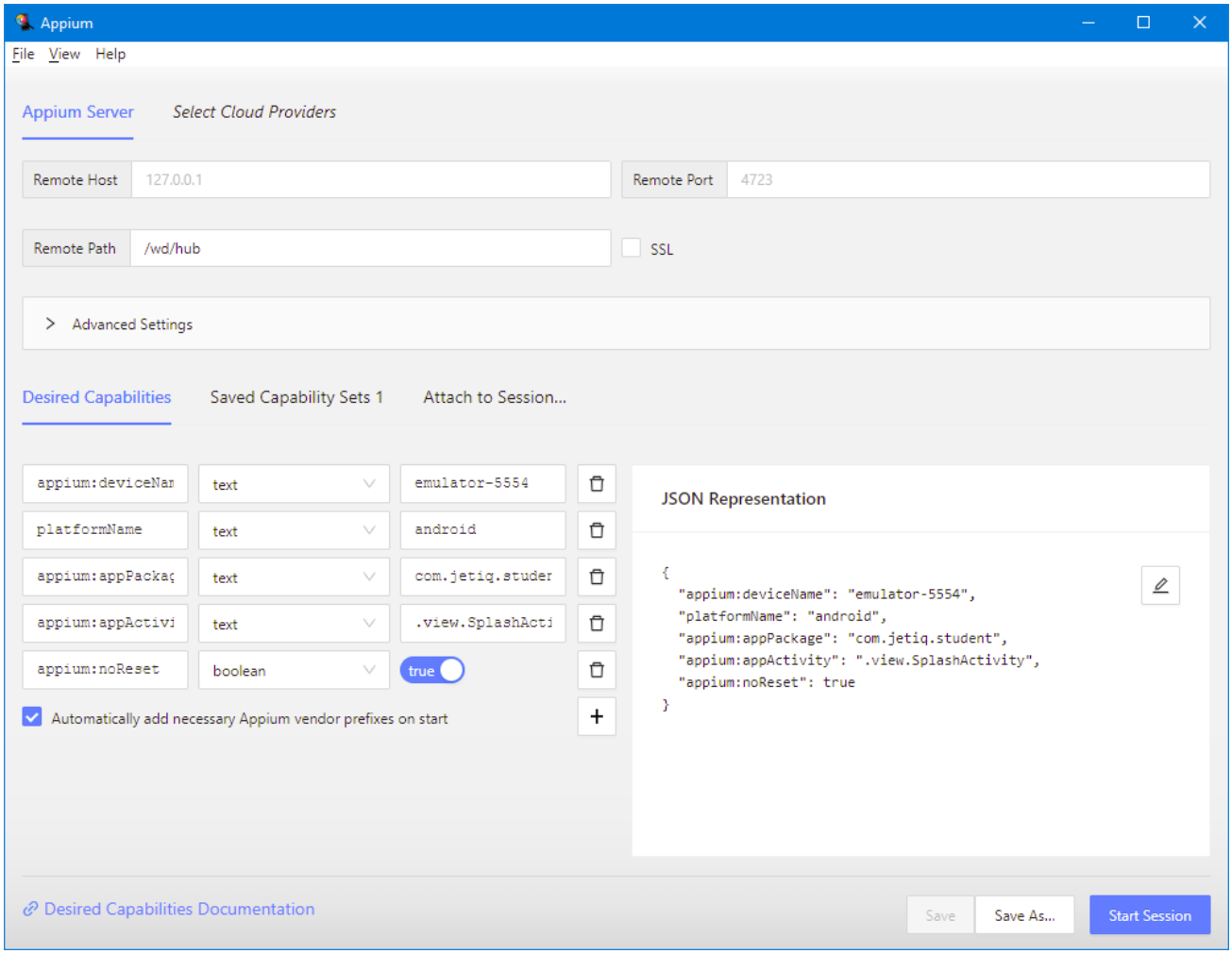

Рисунок 4.12 – Вікно налаштування Appium Inspector

Налаштування Appium та Appium Inspector, повинні збігатись на етапі налаштування серверу, тому ці поля можна залишити пустими і вони будуть використовувати стандартні значення(рисунок 4.13).

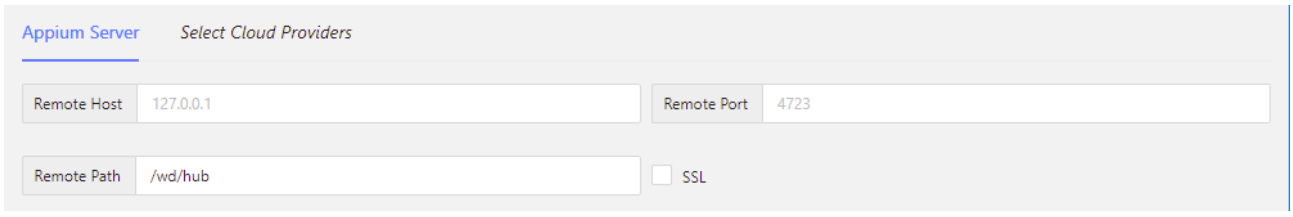

Рисунок 4.13 – Налаштування з’єднання

Обов’язковим налаштуванням інспектора є налаштування пристрою із яким відбувається з’єднання, вони заповняються у поля або у форматі JSON.

Також, за необхідності, можна зберігати вже підготовлені набори налаштувань для пристрою.

Приклад налаштувань зображено на рисунку 4.14.

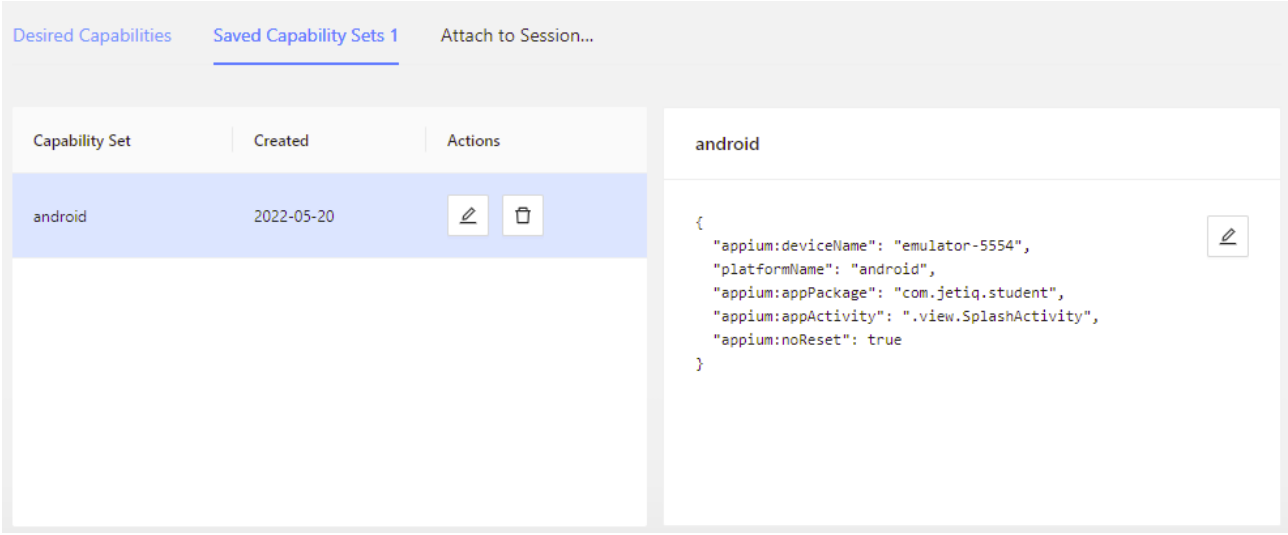

Рисунок 4.14 – Приклад використаног набору налаштувань

Отже, розроблено інструкцію користувача, описано кроки налаштування середовища та інструментів для використання фреймворку для тестування мобільного додатку JetIQ.

4.4 Висновки

1. Тестування фреймворку показало повну його працездатність та відповідність завданню на роботу.

2. Розроблену керівництво програміста, яке забезпечує експлуатацію та подальший розвиток додатку на умовах GNU General Public License.

ВИСНОВКИ

У роботі розроблено фреймворк для тестування мобільного додатку JetIQ та проведено його тестування. Основні результати, отримані при виконанні роботи такі:

- для фреймворку обрано тестування UI, оскільки воно забезпечує перевірку функціоналу мобільного додатку та є більш наглядним у порівнянні з API-тестуванням;
- аналіз методів розробки фреймворків для тестування мобільних додатків показав, що переваги має BDD-методологія розробки програмного забезпечення, яка спрощує розробку тестів за рахунок побудови сценаріїв поведінки;
- аналіз інструментів реалізації BDD-методології показав, що найбільші переваги має спеціалізований фреймворк Cucumber, мова документації - Gherkin, яка описує поведінку системи (BDD) за допомогою сценарії, а також Appium-сервер для реалізації з'єднання з мобільним пристроєм;
- для розробки фреймворку тестування мобільного додатку JetIQ обрано середовище розробки IntelliJ IDEA, середовище для проведення емуляції пристрою на базі операційної системи Android – Android Studio та мову програмування Java, оскільки ці інструменти в повній мірі відповідають принципам RAD;
- вперше запропоновано метод автоматизації тестування мобільного додатку JetIQ, особливість якого полягає у використанні фреймворку, що дозволяє підвищити швидкість тестування мобільного додатку JetIQ в 5-10 разів у порівнянні з ручним тестуванням;
- подальшого розвитку отримав метод розробки фреймворку для тестування мобільного додатку JetIQ, у якому, на відміну від

існуючих, використано BDD-методологію розробки програмного забезпечення, що спростило розробку тестів за рахунок побудови сценаріїв поведінки;

- мовою програмування Java розроблено фреймворк для тестування мобільного додатку JetIQ та сценарії тестів з використанням мови документації Gherkin та фреймворку Cucumber;
- тестування фреймворку показало повну його працездатність та відповідність завданню на роботу;
- розроблено керівництво програміста, яке забезпечує експлуатацію та подальший розвиток додатку на умовах GNU General Public License.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Олійник І.А. Розробка фреймворку для тестування мобільного додатку JetIQ [Електронний ресурс] / Майданюк В.П., Олійник І. А., Коваль Л.Г. // LI Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії, Вінниця: Вінницький Національний Технічний Університет. – 2022. URL: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2022/paper/view/15825

2. Види тестувань. [Електронний ресурс] URL: https://uk.myservername .com/types-automation-testing#Automation_Tools

3. Проектування ПЗ [Електронний ресурс] URL: https://habr .com/ru/post/459620/

4. Життєвий цикл тестування ПЗ [Електронний ресурс] URL: https://www.quality-assurance-group.com/shpargalka-dlya-qa-z-testuvannya-100-najposhyrenishyh-zapytan-na-intervyu/

5. Behavior Driven Development [Електронний ресурс] URL: https://scrumtrek .ru/blog/product-management/5615/behavior-driven-development/

6. Cucumber [Електронний ресурс] URL: https://cucumber.io

7. Gherkin [Електронний ресурс] URL: https://www.linkedin.com/pulse/что-такое-gherkin-айнура-насибова/?originalSubdomain=ru

8. Appium [Електронний ресурс] URL: https://habr.com/ru/post/488482/

9. Java [Електронний ресурс] URL: https://ru.wikipedia.org/wiki/Java

10. Eclipse IDE [Електронний ресурс] URL: https://uk.wikipedia.org/wiki /Eclipse

11. NetBeans IDE [Електронний ресурс] URL: https://ru.wikipedia .org/wiki/NetBeans

12. IntelliJ IDEA [Електронний ресурс] URL: https://uk.wikipedia.org /wiki/IntelliJ_IDEA

13. Android Studio [Електронний ресурс] URL: https://uk.wikipedia.org /wiki/Android_Studio

ДОДАТОК А
(обов’язковий)

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп’ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедрою ПЗ
д.т.н., проф.
_____ О. Н. Романюк
"25" березня 2022 р.

Технічне завдання
на бакалаврську дипломну роботу «Розробка фреймворку для тестування мобільного додатку JetIQ» за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:
_____ к.т.н., доцент Л.Г. Коваль
"__" _____ 2022 р.
Виконав:
_____ студент гр. 2ПІ-18б І.А. Олійник
"__" _____ 2022 р.

Вінниця – 2022 року

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка фреймворку для тестування мобільного додатку JetIQ».

Галузь застосування – тестування, проведення автоматизованого тестування.

2. Підстава для розробки.

Підставою для виконання бакалаврської дипломної роботи (БДР) є індивідуальне завдання на БДР та наказ № 66 від 24 березня 2022 р. ректора ВНТУ про закріплення тем БДР.

3. Мета та призначення розробки.

Метою бакалаврської роботи є оптимізація автоматизованого тестування мобільного додатку JetIQ.

Основними задачами дослідження є:

- обґрунтування вибору типу тестування та підходів до розробки фреймворку для тестування мобільного додатку;
- аналіз технологій, підходів та систем для реалізації автоматизованого тестування;
- розробка фреймворку для тестування мобільного додатку JetIQ;
- проведення тестування мобільного додатку JetIQ за допомогою розробленого фреймворку.

Призначення роботи – розробка та програмна реалізація програмного забезпечення для тестування мобільного додатку.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БДР.

1. Види тестувань. [Електронний ресурс] URL: https://uk.myservername.com/types-automation-testing#Automation_Tools

2. Проектування ПЗ [Електронний ресурс] URL: https://habr.com/ru/post/459620/

3. Життєвий цикл тестування ПЗ [Електронний ресурс] URL: https://www.quality-assurance-group.com/shpargalka-dlya-qa-z-testuvannya-100-najposhyrenishyh-zapytan-na-intervyu/

4. Behavior Driven Development [Електронний ресурс] URL: https://scrumtrek .ru/blog/product-management/5615/behavior-driven-development/

5. Технічні вимоги

Модель розробки – ітеративна; вхідні дані – описані сценарії тестування, налаштоване середовище емуляції мобільного додатку; вихідні дані – логи виконання тестування; середовище розробки – IntelliJ IDEA; мова програмування – JAVA.

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

1. Пояснювальна записка до БДР;

2. Технічне завдання;

3. Лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів дипломного проекту	Термін виконання етапів проекту (роботи)
1	Аналіз предметної області	25.03.22 – 08.04.22
2	Аналіз сучасного стану автоматизованого тестування	11.04.22 – 21.04.22
3	Розробка архітектури програми	22.04.22– 30.04.22
4	Програмна реалізація додатку	02.05.22– 13.05.22
5	Тестування роботи додатку	16.05.22– 24.05.22
6	Оформлення матеріалів до захисту БДР	27.05.22– 08.06.22

10. Порядок контролю та прийняття

Порядок контролю і приймання роботи регламентується відповідними документами ВНТУ і державними стандартами.

ДОДАТОК Б
(обов’язковий)

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка фреймворку для тестування мобільного додатку JetIQ

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: Коваль Л.Г.

Оригінальність	81,4 %
Схожість	18,6 %

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Олійник І.А.

Керівник роботи _____ Коваль Л.Г.

ДОДАТОК В
(довідниковий)

Лістинг програмного коду розробки фреймворку для тестування мобільного додатку JetIQ

Login.feature

```
Feature: Log in
  As a user
  I want to be able to log in into my account
  So that i can be sure that functional log in form works correctly
  Scenario Outline: Verify log-in workflow with login
    Given  User enter login into login field
    When User enter '<correctPassword>' into password field
    And User clicks on 'sign in' button
    Then User sees 'Auto-filling register' alert
    When User doesn't give permission for auto filling register
    Then User sees personal account
    Examples:
      | correctPassword |
      | 9293            |
  Scenario: Verify log-in workflow with nickname
    Given  User enter nickname into nickname field
    When User enter '9293' into password field
    And User clicks on 'sign in' button
    Then User sees 'Auto-filling register' alert
    When User give permission for auto filling register
    Then User sees personal account
```

Smoke.feature

```
Feature: Smoke
  As a user
  I want to be able to use main program functional
  So that i can be sure that main functional works correctly
  Background:
    Given User perform login into account
  Scenario: Check menu filling
    Given User opens navigation menu
    Then User sees navigation menu
    Then User sees 'profile' navigation button
    Then User sees 'notifications' navigation button
    Then User sees 'schedule' navigation button
    Then User sees 'bell schedule' navigation button
    Then User sees 'session schedule' navigation button
    Then User sees 'absence' navigation button
    Then User sees 'old messenger' navigation button
    Then User sees 'messenger' navigation button
    Then User sees 'materials' navigation button
  Scenario: Check notifications page availability
    Given User opens navigation menu
    Then User sees navigation menu
    Then User sees 'notifications' navigation button
    When User opens 'notifications' page
    Then User sees 'notifications' page
  Scenario: Check error message about notifications is empty
    Given User opens navigation menu
    Then User sees navigation menu
    Then User sees 'notifications' navigation button
    When User opens 'notifications' page
    Then User sees notifications is empty error message
  Scenario: Check schedule page availability
    Given User opens navigation menu
    Then User sees navigation menu
    Then User sees 'schedule' navigation button
    When User opens 'schedule' page
    Then User sees 'schedule' page
```

```
  Scenario Outline: Check message from old messenger
    Given User opens navigation menu
    Then User sees navigation menu
    Then User sees 'old messenger' navigation button
    When User opens 'old messenger' page
    Then User sees 'old messenger' page
    Then User sees author of message with name '<name>'
    Then User sees body of message
    Then User check that body contains text '<text>'
    Examples:
      | name       | text          |
      | Ярмола В.С. | додатку JetIQ |
  Scenario: Check the functionality of the profile
    Given User check faculty information contains text 'faculty'
    Given User check group information contains text 'group'
    Given User check course information contains text 'course'
    Then User sees navigation menu button
    When User opens navigation menu
    Then User sees navigation menu
    When User close navigation menu
    And User clicks on 'materials' button on profile page
    Then User sees 'materials' page
    When User navigate back
    When User clicks on 'schedule' button on profile page
    Then User sees 'schedule' page
    When User navigate back
    And User clicks on 'old messenger' button on profile page
    Then User sees 'notifications' page
```

Hooks.class

```
package stepdefinitions;
import io.appium.java_client.AppiumDriver;
import io.appium.java_client.remote.MobileCapabilityType;
import io.cucumber.java.Before;
import manager.PageFactoryManager;
import org.openqa.selenium.remote.DesiredCapabilities;
import java.net.URL;
import java.time.Duration;
public class Hooks {
    DesiredCapabilities capabilities = new DesiredCapabilities();
    AppiumDriver driver;
    PageFactoryManager pageFactoryManager;
    @Before
    public void testSetup() {
        try {
            capabilities.setCapability(MobileCapabilityType.DEVICE_NAME, "emulator-5554");
            capabilities.setCapability(MobileCapabilityType.PLATFORM_NAME, "ANDROID");
            capabilities.setCapability("appPackage", "com.jetiq.student");
            capabilities.setCapability("appActivity", ".view.SplashActivity");
            capabilities.setCapability("autoGrantPermissions", "true");
            URL url = new URL("http://127.0.0.1:4723/wd/hub");
            driver = new AppiumDriver(url, capabilities);
            driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(5));
            pageFactoryManager = PageFactoryManager.getInstance();
            pageFactoryManager.setDriver(driver);
        } catch (Exception exp) {
            System.out.println("Cause is: " + exp.getCause());
            System.out.println("Message is: " + exp.getMessage());
            exp.printStackTrace();
     }}}
```

LoginStepDefs.class

```
package stepdefinitions;
import io.cucumber.java.en.And;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
import manager.PageFactoryManager;
import org.junit.Assert;
import java.util.logging.Logger;
```

```
public class LoginStepDefs {
    private static final long TIME = 5;
    private static Logger log = Logger.getLogger(Hooks.class.getName());
    @When("User clicks on back button")
    public void userClicksOnBackButton() {
        PageFactoryManager.getInstance().getHomePage().clicksOnBackButton();
    }

    @Given("User perform login into account")
    public void userPerformLogin() {
        PageFactoryManager.getInstance().getLogInPage();
        PageFactoryManager.getInstance().getLogInPage().implicitWait(TIME);
        PageFactoryManager.getInstance().getLogInPage().nicknameFieldIsDisplayed();
        PageFactoryManager.getInstance().getLogInPage().clickOnNicknameField();
        PageFactoryManager.getInstance().getLogInPage().setTextToInputNicknameField("zilper");
        log.info("Nickname was entered..");
        PageFactoryManager.getInstance().getLogInPage().setTextToInputPasswordField("9293");
        log.info("Password was entered..");
        PageFactoryManager.getInstance().getLogInPage().clickOnSignInButton();
        PageFactoryManager.getInstance().getHomePage().clicksOnAutoFillRegisterAlertAcceptButton();
        log.info("Allow permission for auto fill..");
    }

    @Given("User enter nickname into nickname field")
    public void userEnterNicknameIntoNicknameField() {

        PageFactoryManager.getInstance().getLogInPage();
        PageFactoryManager.getInstance().getLogInPage().implicitWait(TIME);
        PageFactoryManager.getInstance().getLogInPage().nicknameFieldIsDisplayed();
        PageFactoryManager.getInstance().getLogInPage().clickOnNicknameField();
        PageFactoryManager.getInstance().getLogInPage().setTextToInputNicknameField("zilper");
        log.info("Nickname was entered..");
    }

    @Given("User enter login into login field")
    public void userEnterLoginIntoLoginField() {
        PageFactoryManager.getInstance().getLogInPage();
        PageFactoryManager.getInstance().getLogInPage().implicitWait(TIME);
        PageFactoryManager.getInstance().getLogInPage().loginFieldIsDisplayed();
        PageFactoryManager.getInstance().getLogInPage().clickOnLoginField();
        PageFactoryManager.getInstance().getLogInPage().setTextToInputLoginField("04-18-394");
        log.info("Login was entered..");
    }

    @When("User enter {string} into password field")
    public void userEnterPasswordIntoPasswordField(String password) {
        PageFactoryManager.getInstance().getLogInPage().setTextToInputPasswordField(password);
        log.info("Password was entered..");
    }

    @And("User clicks on 'sign in' button")
    public void userClicksOnSignInButton() {
        PageFactoryManager.getInstance().getLogInPage().clickOnSignInButton();
    }

    @Then("User sees 'Auto-filling register' alert")
    public void userSeeAutofillRegisterAlert() {
        log.info("Waiting for alert..");
        PageFactoryManager.getInstance().getHomePage();
        PageFactoryManager.getInstance().getHomePage().implicitWait(TIME);
        Assert.assertTrue(PageFactoryManager.getInstance().getHomePage().autoFillRegisterAlertIsDisplayed());
        log.info("Alert is appear..");
    }

    @And("User give permission for auto filling register")
    public void userGivePermissionForAutoFillingRegister() {
        PageFactoryManager.getInstance().getHomePage().implicitWait(TIME);
        PageFactoryManager.getInstance().getHomePage().autoFillRegisterAlertAcceptButtonIsDisplayed();
        PageFactoryManager.getInstance().getHomePage().clicksOnAutoFillRegisterAlertAcceptButton();
        log.info("Allow permission for auto fill..");}
```

```
    @And("User doesn't give permission for auto filling register")
    public void userDoesNotGivePermissionForAutoFillingRegister() {
        PageFactoryManager.getInstance().getHomePage().implicitWait(TIME);
        PageFactoryManager.getInstance().getHomePage().autoFillRegisterAlertDeclineButtonIsDisplayed();
        PageFactoryManager.getInstance().getHomePage().clicksOnAutoFillRegisterAlertDeclineButton();
        log.info("Does not allow permission for auto fill..");
    }
    @Then("User sees personal account")
    public void userSeesPersonalAccount() {
        PageFactoryManager.getInstance().getHomePage().implicitWait(TIME);
        boolean element1 = PageFactoryManager.getInstance().getHomePage().avatarImageIsVisible();
        boolean element2 = PageFactoryManager.getInstance().getHomePage().logoutButtonIsVisible();
        Assert.assertEquals(element1,element2);
        log.info("Personal account is opened..");
    }}
```

SmokeStepDefs.class

```
package stepdefinitions;
import io.cucumber.java.en.And;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
import manager.PageFactoryManager;
import org.junit.Assert;
import java.util.logging.Logger;
public class SmokeStepDefs {
    private static final long TIME = 5;
    private static Logger log = Logger.getLogger(Hooks.class.getName());
    @Given("User opens navigation menu")
    public void userOpensNavigationMenu() {
        PageFactoryManager.getInstance().getHomePage().implicitWait(TIME);
        PageFactoryManager.getInstance().getHomePage().menuButtonIsVisible();
        log.info("Navigation button is present..");
        PageFactoryManager.getInstance().getHomePage().clicksOnMenuButton();
        log.info("Clicks on navigation button..");
    }
    @Then("User sees navigation menu")
    public void userSeesNavigationMenu() {
        PageFactoryManager.getInstance().getHomePage().implicitWait(TIME);
        Assert.assertTrue(PageFactoryManager.getInstance().getHomePage().navigationMenuIsVisible());
        log.info("Navigation menu is appear..");}
    @Then("User sees 'profile' navigation button")
    public void userSeesProfileNavigationButton() {
        PageFactoryManager.getInstance().getHomePage().implicitWait(TIME);
        Assert.assertTrue(PageFactoryManager.getInstance().getHomePage().profileNavigationMenuButtonIsVisible());
        log.info("Profile navigation button is visible..");
    }
    @Then("User sees 'notifications' navigation button")
    public void userSeesNotificationsNavigationButton() {
        PageFactoryManager.getInstance().getHomePage().implicitWait(TIME);
        Assert.assertTrue(PageFactoryManager.getInstance().getHomePage().notificationNavigationMenuButtonIsVisible());
        log.info("Notifications navigation button is visible..");
    }
    @Then("User sees 'schedule' navigation button")
    public void userSeesScheduleNavigationButton() {
        PageFactoryManager.getInstance().getHomePage().implicitWait(TIME);
        Assert.assertTrue(PageFactoryManager.getInstance().getHomePage().scheduleNavigationMenuButtonIsVisible());
        log.info("Schedule navigation button is visible..");
    }
    @Then("User sees 'bell schedule' navigation button")
    public void userSeesBellScheduleNavigationButton()
        PageFactoryManager.getInstance().getHomePage().implicitWait(TIME);
        Assert.assertTrue(PageFactoryManager.getInstance().getHomePage().bellScheduleNavigationMenuButtonIsVisible());
        log.info("Bell schedule navigation button is visible..");
    }
    @Then("User sees 'session schedule' navigation button")
    public void userSeesSessionScheduleNavigationButton() {
        PageFactoryManager.getInstance().getHomePage().implicitWait(TIME);
        Assert.assertTrue(PageFactoryManager.getInstance().getHomePage().sessionScheduleNavigationMenuButtonIsVisible());
        log.info("Session schedule navigation button is visible..");
    }
```

```
@Then("User sees 'absence' navigation button")
public void userSeesAbsenceNavigationButton() {
   PageFactoryManager.getInstance().getHomePage().implicitWait(TIME);
   Assert.assertTrue(PageFactoryManager.getInstance().getHomePage().absenceNavigationMenuButtonIsVisible());
   log.info("Absence navigation button is visible..");
}

@Then("User sees 'old messenger' navigation button")
public void userSeesOldMessengerNavigationButton() {
   PageFactoryManager.getInstance().getHomePage().implicitWait(TIME);
   Assert.assertTrue(PageFactoryManager.getInstance().getHomePage().oldMessengerNavigationMenuButtonIsVisible());
   log.info("Old messenger navigation button is visible..");
}

@Then("User sees 'messenger' navigation button")
public void userSeesMessengerNavigationButton() {
   PageFactoryManager.getInstance().getHomePage().implicitWait(TIME);
   Assert.assertTrue(PageFactoryManager.getInstance().getHomePage().messengerNavigationMenuButtonIsVisible());
   log.info("Messenger navigation button is visible..");
}

@Then("User sees 'materials' navigation button")
public void userSeesMaterialsNavigationButton() {
   PageFactoryManager.getInstance().getHomePage().implicitWait(TIME);
   Assert.assertTrue(PageFactoryManager.getInstance().getHomePage().materialsNavigationMenuButtonIsVisible());
   log.info("Materials navigation button is visible..");
}

@When("User opens 'notifications' page")
public void userOpensNotificationsPage() {
   PageFactoryManager.getInstance().getHomePage().implicitWait(TIME);
   PageFactoryManager.getInstance().getHomePage().clicksOnNotificationsNavigationMenuButton();
   log.info("Notifications page is appear..");
}

@When("User opens 'schedule' page")
public void userOpensSchedulePage() {
   PageFactoryManager.getInstance().getHomePage().implicitWait(TIME);
   PageFactoryManager.getInstance().getHomePage().clicksOnScheduleNavigationMenuButton();
   log.info("Schedule page is appear..");
}

@When("User opens 'bell schedule' page")
public void userOpensBellSchedulePage() {
   PageFactoryManager.getInstance().getHomePage().implicitWait(TIME);
   PageFactoryManager.getInstance().getHomePage().clicksBellScheduleNavigationMenuButton();
   log.info("Bell schedule page is appear..");
}

@When("User opens 'session schedule' page")
public void userOpensSessionSchedulePage() {
   PageFactoryManager.getInstance().getHomePage().implicitWait(TIME);
   PageFactoryManager.getInstance().getHomePage().clicksSessionScheduleNavigationMenuButton();
   log.info("Session schedule page is appear..");
}

@When("User opens 'absence' page")
public void userOpensAbsencePage() {
   PageFactoryManager.getInstance().getHomePage().implicitWait(TIME);
   PageFactoryManager.getInstance().getHomePage().clicksAbsenceNavigationMenuButton();
   log.info("Absence page is appear..");
}
@When("User opens 'old messenger' page")
public void userOpensOldMessengerPage() {
   PageFactoryManager.getInstance().getHomePage().implicitWait(TIME);
   PageFactoryManager.getInstance().getHomePage().clicksOldMessengerNavigationMenuButton();
   log.info("Old messenger page is appear..");

}
```

```
@When("User opens 'messenger' page")
public void userOpensMessengerPage() {
   PageFactoryManager.getInstance().getHomePage().implicitWait(TIME);
   PageFactoryManager.getInstance().getHomePage().clicksMessengerNavigationMenuButton();
   log.info("Messenger page is appear..");
}
@When("User opens 'materials' page")
public void userOpensMaterialsPage() {
   PageFactoryManager.getInstance().getHomePage().implicitWait(TIME);
   PageFactoryManager.getInstance().getHomePage().clicksMaterialsNavigationMenuButton();
   log.info("Materials page is appear..");
}
@Then("User sees 'notifications' page")
public void userSeesNotificationsPage() {
   PageFactoryManager.getInstance().getNotificationsPage().implicitWait(TIME);
   Assert.assertTrue(PageFactoryManager.getInstance().getNotificationsPage().notificationListIsVisible());
   log.info("Notifications page is opened..");
}
@Then("User sees notifications is empty error message")
public void userSeesNotificationsIsEmptyErrorMessage() {
   PageFactoryManager.getInstance().getNotificationsPage().implicitWait(TIME);
   Assert.assertTrue(PageFactoryManager.getInstance().getNotificationsPage().errorMessageOnNotificationPageIsVisible());
   log.info("Notification error is present on page..");
}
@Then("User sees 'schedule' page")
public void userSeesSchedulePage() {
   PageFactoryManager.getInstance().getSchedulePage().implicitWait(TIME);
   Assert.assertTrue(PageFactoryManager.getInstance().getSchedulePage().schedulePageIsVisible());
   log.info("Schedule page is opened..");
}
@Then("User sees 'old messenger' page")
public void userSeesOldMessengerPage() {
   PageFactoryManager.getInstance().getOldMessengerPage().implicitWait(TIME);
   Assert.assertTrue(PageFactoryManager.getInstance().getOldMessengerPage().messageListIsVisible());
   log.info("Old messenger page is opened..");
}
@Then("User sees author of message with name {string}")
public void userSeesAuthorOfMessageWithNameName(String name) {
   PageFactoryManager.getInstance().getOldMessengerPage().implicitWait(TIME);
   PageFactoryManager.getInstance().getOldMessengerPage().messageAuthorIsVisible();
   log.info("Author name is visible..");
   Assert.assertTrue(PageFactoryManager.getInstance().getOldMessengerPage().messageAuthorContainsText(name));
   log.info("Author name is equal to expected..");
}
@Then("User sees body of message")
public void userSeesBodyOfMessage() {
   PageFactoryManager.getInstance().getOldMessengerPage().implicitWait(TIME);
   PageFactoryManager.getInstance().getOldMessengerPage().messageBodyIsVisible();
   log.info("Body of message is visible..");
}
@Then("User check that body contains text {string}")
public void userCheckThatBodyContainsTextText(String text) {
   PageFactoryManager.getInstance().getOldMessengerPage().implicitWait(TIME);
   Assert.assertTrue(PageFactoryManager.getInstance().getOldMessengerPage().messageBodyContainsText(text));
   log.info("Body contains expected value..");
}
@Then("User sees navigation menu button")
public void userSeesNavigationMenuButton() {
   PageFactoryManager.getInstance().getHomePage().implicitWait(TIME);
   Assert.assertTrue(PageFactoryManager.getInstance().getHomePage().menuButtonIsVisible());
   log.info("Menu button is present on page..");
}
@Given("User check faculty information contains text {string}")
public void userCheckFacultyInformationContainsTextFaculty(String facultyName) {
   PageFactoryManager.getInstance().getHomePage().implicitWait(TIME);
   Assert.assertTrue(PageFactoryManager.getInstance().getHomePage().facultyElementContainsCorrectValue(facultyName));
   log.info("Profile has correct faculty..");

}
```

```
    @Given("User check group information contains text {string}")
    public void userCheckGroupInformationContainsTextGroup(String groupNumber) {
        PageFactoryManager.getInstance().getHomePage().implicitWait(TIME);
        Assert.assertTrue(PageFactoryManager.getInstance().getHomePage().groupElementContainsCorrectValue(groupNumber));
    }
    @Given("User check course information contains text {string}")
    public void userCheckCourseInformationContainsTextCourse(String courseNumber) {
        PageFactoryManager.getInstance().getHomePage().implicitWait(TIME);
        Assert.assertTrue(PageFactoryManager.getInstance().getHomePage().courseElementContainsCorrectValue(courseNumber));
    }
    @When("User close navigation menu")
    public void userCloseNavigationMenu() {
        PageFactoryManager.getInstance().getHomePage().implicitWait(TIME);
        PageFactoryManager.getInstance().getHomePage().closeNavigationMenu();
    }
    @When("User navigate back")
    public void userNavigateBack() {
        PageFactoryManager.getInstance().getHomePage().implicitWait(TIME);
        PageFactoryManager.getInstance().getHomePage().clicksOnBackButton();
    }
    @Then("User sees 'materials' page")
    public void userSeesMaterialsPage() {
        PageFactoryManager.getInstance().getMaterialsPage().implicitWait(TIME);
        Assert.assertTrue(PageFactoryManager.getInstance().getMaterialsPage().materialsListIsVisible());
    }
    @And("User clicks on 'materials' button on profile page")
    public void userClicksOnMaterialsButtonOnProfilePage() {
        PageFactoryManager.getInstance().getHomePage().implicitWait(TIME);
        PageFactoryManager.getInstance().getHomePage().clicksMaterialsButtonOnProfilePage();
    }
    @When("User clicks on 'schedule' button on profile page")
    public void userClicksOnScheduleButtonOnProfilePage() {
        PageFactoryManager.getInstance().getHomePage().implicitWait(TIME);
        PageFactoryManager.getInstance().getHomePage().clicksScheduleButtonOnProfilePage();
    }
    @And("User clicks on 'old messenger' button on profile page")
    public void userClicksOnNotificationsButtonOnProfilePage() {
        PageFactoryManager.getInstance().getHomePage().implicitWait(TIME);
        PageFactoryManager.getInstance().getHomePage().clicksOldMessengerButtonOnProfilePage();
}}
```

PageFactoryManager.class

```
package manager;
import io.appium.java_client.AppiumDriver;
import pages.*;
public class PageFactoryManager {
    private static volatile PageFactoryManager instance;
    private AppiumDriver driver;
    public AppiumDriver getDriver() { return driver; }
    public PageFactoryManager() {}
    public static PageFactoryManager getInstance() {
        if (instance == null) {
            instance = new PageFactoryManager();
        }return instance;
    }
    public void setDriver(AppiumDriver driver) {
        this.driver = driver;
    }
    public LogInPage getLogInPage() {
        return new LogInPage(driver);
    }
    public HomePage getHomePage() {
        return new HomePage(driver);
    }
    public NotificationsPage getNotificationsPage() { return new NotificationsPage(driver); }
    public SchedulePage getSchedulePage() { return new SchedulePage(driver); }
    public OldMessengerPage getOldMessengerPage() { return new OldMessengerPage(driver); }
    public MaterialsPage getMaterialsPage() { return new MaterialsPage(driver); }

}
```

BasePage.class

```
package pages;
import io.appium.java_client.AppiumDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;
import java.time.Duration;
public class BasePage {
    public AppiumDriver driver;
    @FindBy(className = "android.widget.ImageButton")
    WebElement backButton;
    public BasePage(AppiumDriver driver) {
        this.driver = driver;
        PageFactory.initElements(driver, this);
    }
    public void implicitWait(final long timeout) {
        driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(timeout));
    }
    public void clicksOnBackButton() { backButton.click(); }
}
```

HomePage.class

```
package pages;
import io.appium.java_client.AppiumDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
public class HomePage extends BasePage {
    @FindBy(id = "android:id/alertTitle")
    WebElement autoFillRegisterAlert;
    @FindBy(id = "android:id/button1")
    WebElement autoFillRegisterAcceptButton;
    @FindBy(id = "android:id/button2")
    WebElement autoFillRegisterDeclineButton;
    @FindBy(id = "action_exit")
    WebElement logoutButton;
    @FindBy(id = "img")
    WebElement avatarImage;
    @FindBy(className = "android.widget.ImageButton")
    WebElement menuButton;
    @FindBy(id = "fragment_navigation_drawer")
    WebElement navigationMenu;
    @FindBy(id = "profile")
    WebElement profileNavigationMenuButton;
    @FindBy(id = "notifications")
    WebElement notificationsNavigationMenuButton;
    @FindBy(id = "schedule")
    WebElement scheduleNavigationMenuButton;
    @FindBy(id = "bell_schedule")
    WebElement bellScheduleNavigationMenuButton;
    @FindBy(id = "session_schedule")
    WebElement sessionScheduleNavigationMenuButton;
    @FindBy(id = "absence")
    WebElement absenceNavigationMenuButton;
    @FindBy(id = "old_messenger")
    WebElement oldMessengerNavigationMenuButton;
    @FindBy(id = "messenger")
    WebElement messengerNavigationMenuButton;
    @FindBy(id = "materials")
    WebElement materialsNavigationMenuButton;
    @FindBy(id = "check_in")
    WebElement checkIn;
    @FindBy(id = "tvFaculty")
    WebElement faculty;
    @FindBy(id = "tvCourse")
    WebElement course;
    @FindBy(id = "tvGroup")
    WebElement group;
    @FindBy(id = "bMaterials")
    WebElement materialsButton;
```

```
    @FindBy(id = "bSchedule")
    WebElement scheduleButton;
    @FindBy(id = "bMess")
    WebElement oldMessengerButton;
    public HomePage(AppiumDriver driver) {
        super(driver);
    }
    public boolean autoFillRegisterAlertIsDisplayed() { return autoFillRegisterAlert.isDisplayed(); }
    public boolean autoFillRegisterAlertAcceptButtonIsDisplayed() { return autoFillRegisterAcceptButton.isDisplayed(); }
    public boolean autoFillRegisterAlertDeclineButtonIsDisplayed() { return autoFillRegisterDeclineButton.isDisplayed(); }
    public boolean logoutButtonIsVisible() { return logoutButton.isDisplayed(); }
    public boolean avatarImageIsVisible() { return avatarImage.isDisplayed(); }
    public boolean menuButtonIsVisible() { return menuButton.isDisplayed(); }
    public boolean navigationMenuIsVisible() { return navigationMenu.isDisplayed(); }
    public boolean profileNavigationMenuButtonIsVisible() { return profileNavigationMenuButton.isDisplayed(); }
    public boolean notificationNavigationMenuButtonIsVisible() { return notificationsNavigationMenuButton.isDisplayed(); }
    public boolean scheduleNavigationMenuButtonIsVisible() { return scheduleNavigationMenuButton.isDisplayed(); }
    public boolean bellScheduleNavigationMenuButtonIsVisible() { return bellScheduleNavigationMenuButton.isDisplayed(); }
    public boolean sessionScheduleNavigationMenuButtonIsVisible() { return
sessionScheduleNavigationMenuButton.isDisplayed();}
    public boolean absenceNavigationMenuButtonIsVisible() { return absenceNavigationMenuButton.isDisplayed(); }
    public boolean oldMessengerNavigationMenuButtonIsVisible() { return oldMessengerNavigationMenuButton.isDisplayed(); }
    public boolean messengerNavigationMenuButtonIsVisible() { return messengerNavigationMenuButton.isDisplayed(); }
    public boolean materialsNavigationMenuButtonIsVisible() { return materialsNavigationMenuButton.isDisplayed(); }
    public boolean materialsButtonOnProfilePageIsVisible() { return materialsButton.isDisplayed(); }
    public boolean scheduleButtonOnProfilePageIsVisible() { return scheduleButton.isDisplayed(); }
    public boolean oldMessengerButtonOnProfilePageIsVisible() { return oldMessengerButton.isDisplayed(); }
    public boolean checkInElementIsVisible() { return checkIn.isDisplayed(); }
    public boolean facultyElementContainsCorrectValue(String facultyName) { return faculty.getText().contains(facultyName); }
    public boolean groupElementContainsCorrectValue(String groupNumber) { return group.getText().contains(groupNumber); }
    public boolean courseElementContainsCorrectValue(String courseNumber) { return course.getText().contains(courseNumber); }
    public void clicksOnAutoFillRegisterAlertAcceptButton() { autoFillRegisterAcceptButton.click(); }
    public void clicksOnAutoFillRegisterAlertDeclineButton() { autoFillRegisterDeclineButton.click(); }
    public void clicksOnLogoutButton() { logoutButton.click(); }
    public void clicksOnMenuButton() { menuButton.click(); }
    public void clicksOnNotificationsNavigationMenuButton() { notificationsNavigationMenuButton.click(); }
    public void clicksOnScheduleNavigationMenuButton() { scheduleNavigationMenuButton.click(); }
    public void clicksBellScheduleNavigationMenuButton() { bellScheduleNavigationMenuButton.click(); }
    public void clicksSessionScheduleNavigationMenuButton() { sessionScheduleNavigationMenuButton.click(); }
    public void clicksAbsenceNavigationMenuButton() { absenceNavigationMenuButton.click(); }
    public void clicksOldMessengerNavigationMenuButton() { oldMessengerNavigationMenuButton.click(); }
    public void clicksMessengerNavigationMenuButton() { messengerNavigationMenuButton.click(); }
    public void clicksMaterialsNavigationMenuButton() { materialsNavigationMenuButton.click(); }
    public void closeNavigationMenu() { profileNavigationMenuButton.click(); }
    public void clicksMaterialsButtonOnProfilePage() { materialsButton.click(); }
    public void clicksScheduleButtonOnProfilePage() { scheduleButton.click(); }
    public void clicksOldMessengerButtonOnProfilePage() { oldMessengerButton.click(); }
}
```

LogInPage.class

```
package pages;
import io.appium.java_client.AppiumDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
public class LogInPage extends BasePage {
    @FindBy(id = "nik")
    WebElement nicknameField;
    @FindBy(id = "login")
    WebElement loginField;
    @FindBy(id = "password")
    WebElement inputPasswordField;
    @FindBy(id = "sign_in")
    WebElement signInButton;
    public LogInPage(AppiumDriver driver) { super(driver); }
    public boolean nicknameFieldIsDisplayed() { return nicknameField.isDisplayed();}
    public boolean loginFieldIsDisplayed() { return loginField.isDisplayed();}
    public void clickOnNicknameField() { nicknameField.click(); }
    public void clickOnLoginField() { loginField.click(); }
    public void setTextToInputNicknameField(final String email){
        nicknameField.sendKeys(email); }
```

```
    public void setTextToInputLoginField(final String email){
        loginField.sendKeys(email);
    }
    public void setTextToInputPasswordField(final String password){
        inputPasswordField.sendKeys(password);
    }
    public void clickOnSignInButton(){ signInButton.click(); }
}

package pages;
import io.appium.java_client.AppiumDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
public class MaterialsPage extends BasePage{
    @FindBy(id = "materialsListView")
    WebElement materialsList;
    public MaterialsPage(AppiumDriver driver) {super(driver);}
    public boolean materialsListIsVisible() { return materialsList.isDisplayed(); }
}
package pages;
import io.appium.java_client.AppiumDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
public class NotificationsPage extends BasePage{
    @FindBy(id = "notificationListView")
    WebElement notificationList;
    @FindBy(id = "err_message")
    WebElement errorMessage;
    public NotificationsPage(AppiumDriver driver) { super(driver); }
    public boolean notificationListIsVisible() { return notificationList.isDisplayed(); }
    public boolean errorMessageOnNotificationPageIsVisible() { return errorMessage.isDisplayed(); }
}
package pages;
import io.appium.java_client.AppiumDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
public class OldMessengerPage extends BasePage{
    @FindBy(id = "author")
    WebElement messageAuthor;
    @FindBy(id = "body")
    WebElement messageBody;
    @FindBy(id = "messageListView")
    WebElement messageList;
    public OldMessengerPage(AppiumDriver driver) {super(driver);}
    public boolean messageListIsVisible() { return messageList.isDisplayed(); }
    public boolean messageAuthorIsVisible() { return messageAuthor.isDisplayed(); }
    public boolean messageAuthorContainsText(String name) { return messageAuthor.getText().contains(name); }
    public boolean messageBodyIsVisible() { return messageBody.isDisplayed(); }
    public boolean messageBodyContainsText( String text) { return messageBody.getText().contains(text); }
}
package pages;
import io.appium.java_client.AppiumDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
public class SchedulePage extends BasePage{
    @FindBy(id = "email_login_form")
    WebElement schedule;
    @FindBy(id = "calendar")
    WebElement calendar;
    public SchedulePage(AppiumDriver driver) { super(driver); }
    public boolean schedulePageIsVisible() { return schedule.isDisplayed(); }

}
```

ДОДАТОК Г
(обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА ФРЕЙМВОРКУ ДЛЯ ТЕСТУВАННЯ МОБІЛЬНОГО ДОДАТКУ JETIQ

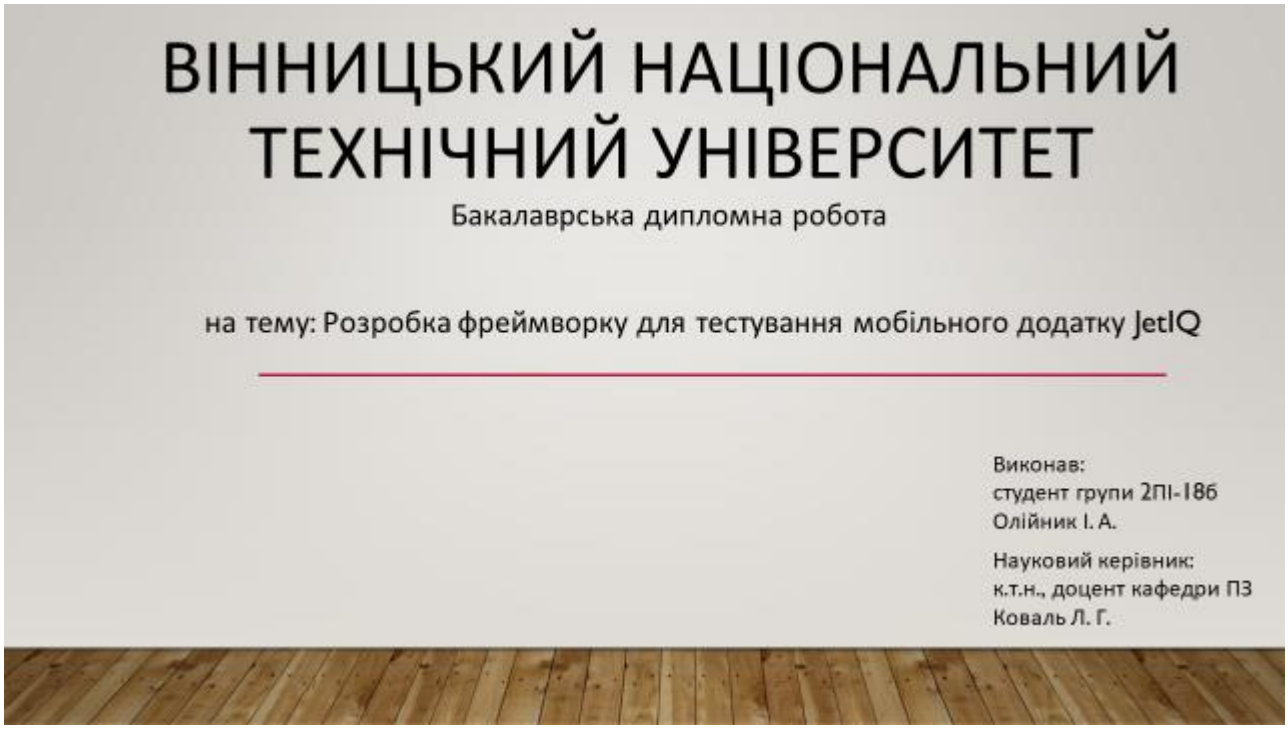

Рисунок Г.1 – Титульний слайд

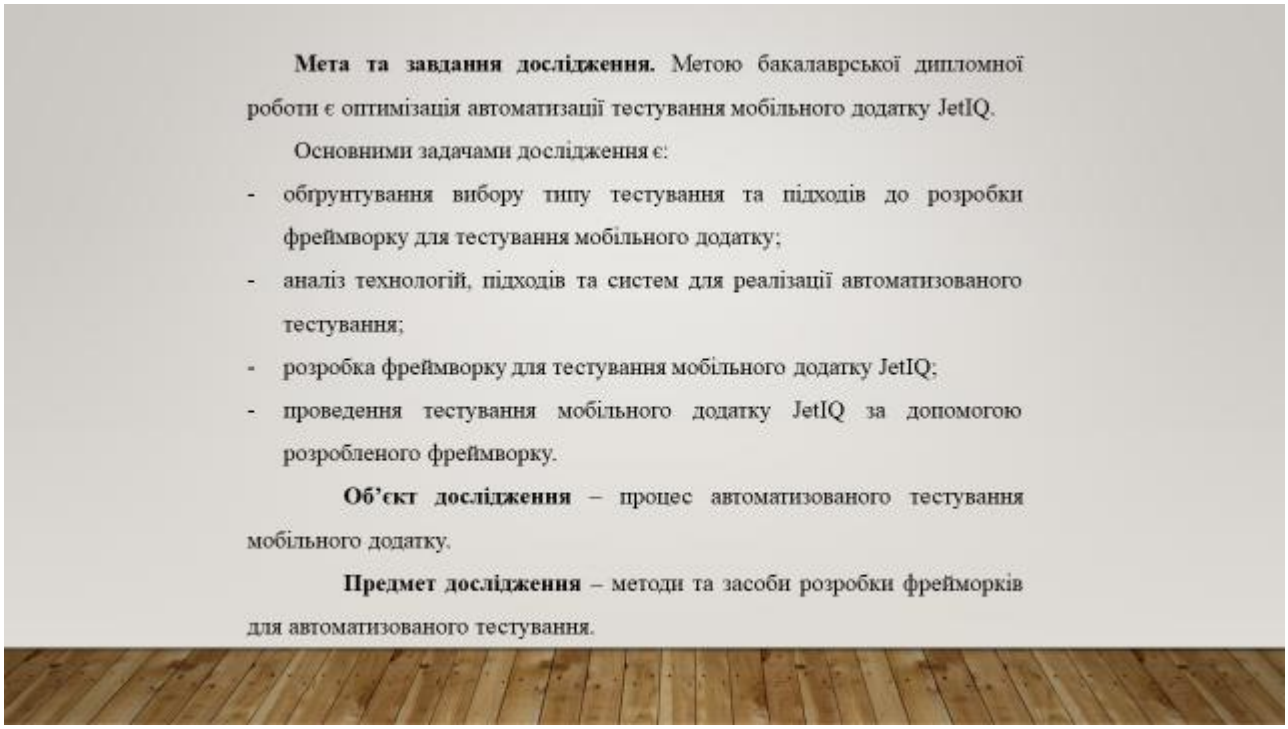

Рисунок Г.2 – Цілі дослідження

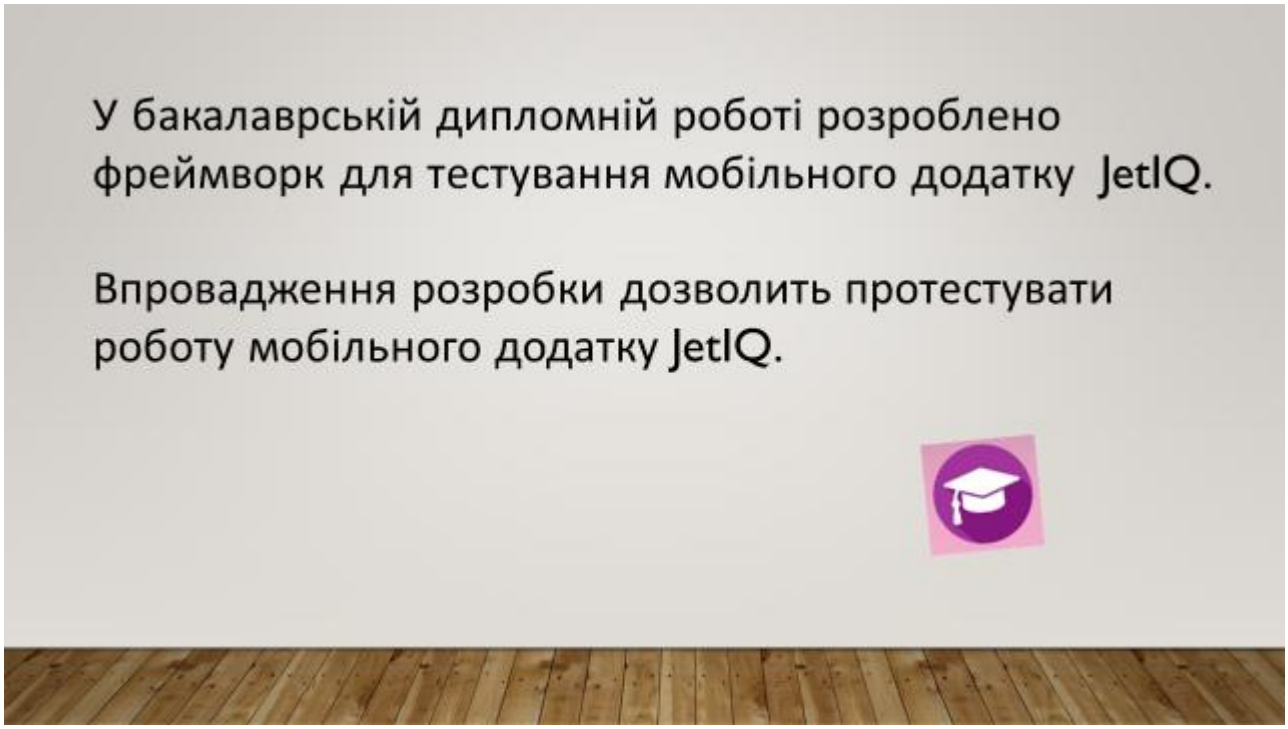

Рисунок Г.3 – Впровадження розробки

Рисунок Г.4 – Діаграма класів

Рисунок Г.5 – Блок-схема створення тестів

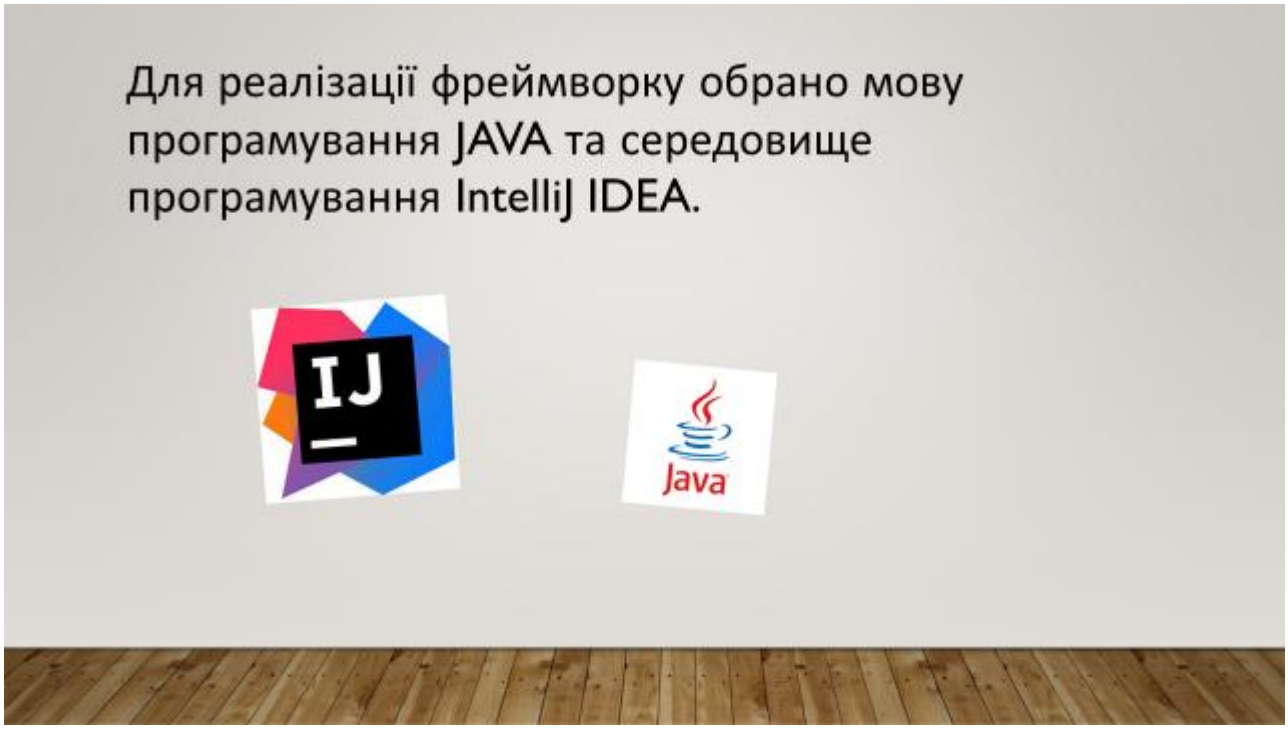

Рисунок Г.6 – Середовище та мова розробки

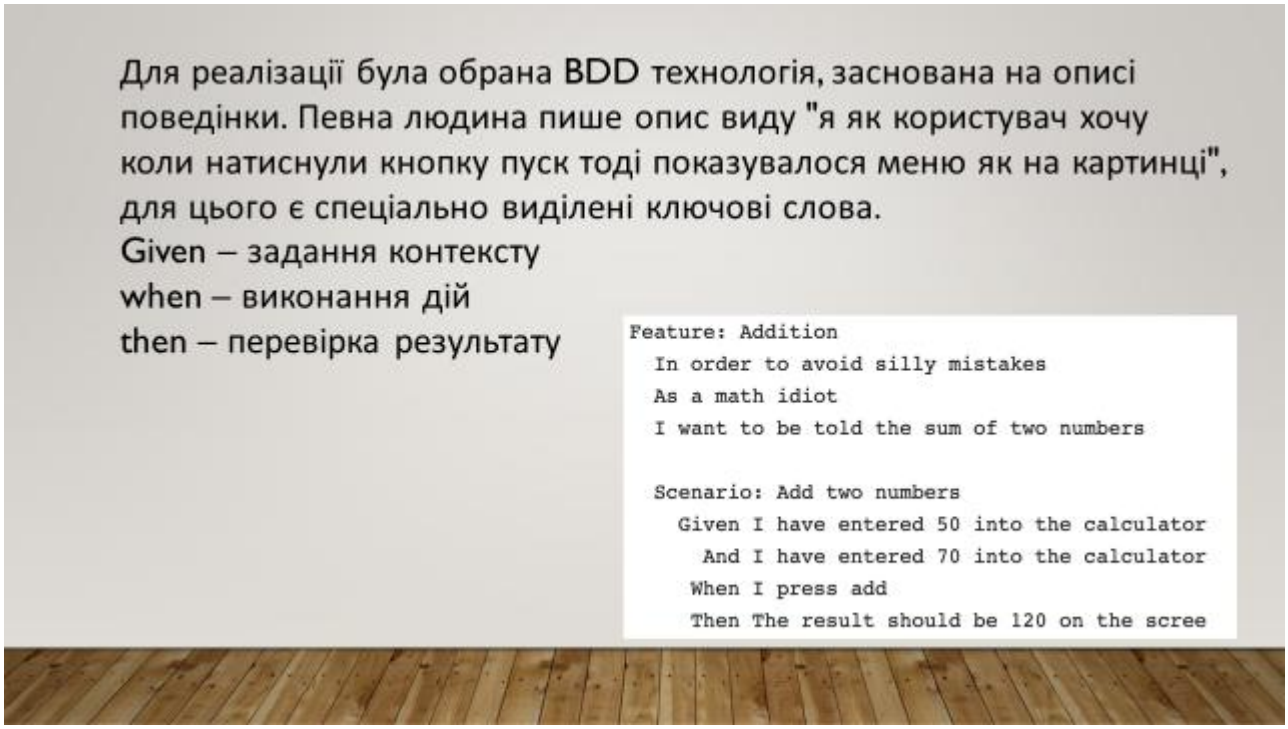

Рисунок Г.7 – Методологія розробки

Рисунок Г.8 – Середовище емуляції Android пристрою

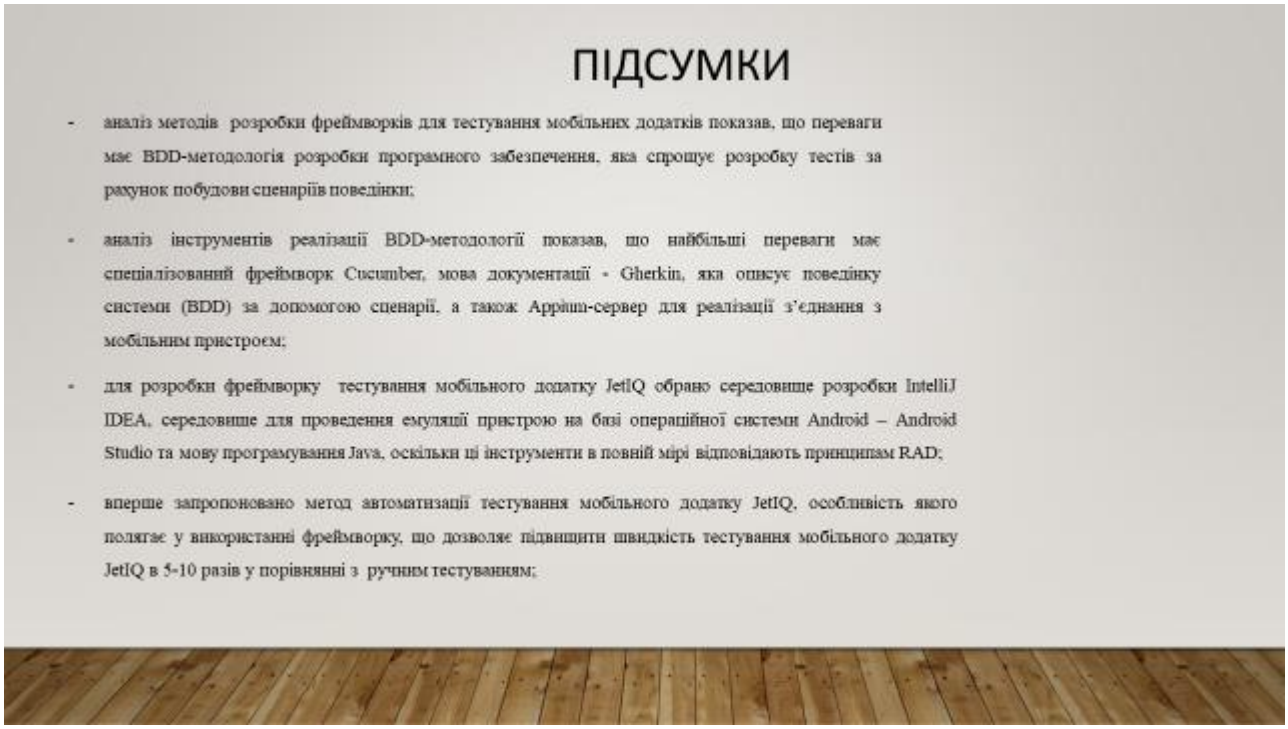

Рисунок Г.9 – Підсумки

Рисунок Г.10 – Фінальний слайд