Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Факультет інформатики
Кафедра інформатики

Створення інструментів для автоматичного тестування графічного інтерфейсу для iOS/iPadOS

Текстова частина до кваліфікаційної роботи
за спеціальністю «Інженерія програмного забезпечення» - 121

Виконав: студент 4-го року навчання,

Освітньої програми «Інженерія програмного забезпечення», 121

Гетьман Дмитро Андрійович

Керівник Франків О.О., _____

Старший викладач

Рецензент _____
(прізвище та ініціали)

Секретар ЕК
_____
«____» _____ 2024 р.

Київ – 2024

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Факультет інформатики
Кафедра інформатики

ЗАТВЕРДЖУЮ
Викладач кафедри інформатики ,
канд. фіз-мат. наук, доц. _____ Гороховський С.С.
(підпис)
„_____”_____2023р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на кваліфікаційну роботу
студенту Гетьману Дмитру Андрійовичу
факультету інформатики 4 курсу бакалаврської програми
ТЕМА: Створення інструментів для автоматичного тестування графічного інтерфейсу для iOS/iPadOS

Зміст ТЧ до кваліфікаційної роботи:
Індивідуальне завдання
Вступ
Огляд теоретичного матеріалу і здійснення дослідження
Висновки
Список посилань

Дата видачі „___” _____ 2023 р. Керівник_____
(підпис)
Завдання отримав _____
(підпис)

Тема: **Створення інструментів для автоматичного тестування графічного інтерфейсу для iOS/iPadOS**

№ п/п	Назва етапу кваліфікаційної роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на кваліфікаційної роботу.	23.09.2023	
2.	Огляд технічної літератури за темою роботи.	22.10. 2023	
3.	Виконати аналіз сучасних рішень.	23.11.2023	
4.	Розробка алгоритму створення інструменту для тестування адаптивності.	27.12.2023	
5.	Реалізація розробленого алгоритму.	15.02. 2024	
6.	Написання пояснювальної роботи.	01.04.2024	
7.	Аналіз отриманих результатів з керівником.	18.04. 2024	
8.	Корегування роботи.	01.05. 2024	
9.	Створення слайдів для доповіді та написання доповіді.	08.05.2024	
10.	Аналіз отриманих результатів з керівником, написання доповіді та попередній захист кваліфікаційної роботи.	13.05.2024	
11.	Корегування роботи за результатами попереднього захисту.	18.05.2024	
12.	Остаточне оформлення пояснювальної роботи та слайдів.	20.05.2024	
13.	Захист кваліфікаційної роботи (проекту)	27.05.2024	

Студент _____

Керівник _____

“_____” _____

Зміст

АНОТАЦІЯ **6**

ВСТУП **7**

РОЗДІЛ 1. ПРОЕКТУВАННЯ ТА ТЕСТУВАННЯ ГРФІЧНОГО ІНТЕРФЕЙСУ **9**

1.1 Figma ***9***

1.1.1 Огляд можливостей та переваг користування Figma для дизайну і прототипування 9

1.1.2 Аналіз впливу Figma на процес розробки мобільних додатків. 11

1.2 CucumberSwift ***13***

1.2.1 Огляд методів тестування мобільних додатків. 13

1.2.2 Роль CucumberSwift у реалізації автоматизованих тестів 14

1.3 Fastlane ***17***

1.3.1 Переваги використання Fastlane у процесі розробки мобільних додатків. 17

1.3.2 Огляд основних функцій та інструментів Fastlane 19

1.4 Висновки ***20***

РОЗДІЛ 2. АВТОМАТИЗАЦІЯ ТЕСТУВАННЯ UI **22**

2.1 Figma ***22***

2.1.1 Візуалізація тестових кейсів 22

2.1.2 Колаборативний процес 22

2.1.3 Тестування на ранніх етапах розробки 23

2.1.4 Ефективне порівняння з дійсним інтерфейсом 24

2.2 CucumberSwift ***24***

2.2.1 BDD (Behavior-Driven Development) підхід 24

2.2.2 Легкість читання та зміни тестів 25

2.2.3 Автоматизація тестування інтерфейсу користувача 26

2.2.4 Інтеграція з іншими інструментами 27

2.3 Fastlane ***27***

2.3.1 Автоматизація тестування 27

2.3.2 Інтеграція з Xcode та XCTest 28

2.3.3 Послідовність тестування 29

2.3.4 Простота налаштування та використання 29

2.3.5 Інтеграція з іншими інструментами 30

2.3.6 Використання Fastlane для запуску процесу тестування 30

2.4 Висновки до розділу ***31***

РОЗДІЛ 3. ОПИС ІМПЛЕМЕНТАЦІЇ ТА АНАЛІЗ РЕЗУЛЬТАТІВ **32**

3.1 Figma *32*
3.1.1 Завантаження та обробка даних з Figma 32
3.1.2 Порівняння макетів з реальним інтерфейсом 33
3.1.3 Створення додатку на основі макетів 34
3.1.4 Тестування відповідності макетів 35
3.1.5 Ефективність процесу тестування 36
3.2 CucumberSwift *37*
3.2.1 Налаштування середовища для розробки тестів 37
3.2.2 Написання тестових сценаріїв 37
3.2.3 Створення самостійних тестів 38
3.2.4 Інтеграція системи порівняння layouts 39
3.2.5 Виконання тестових сценаріїв 40
3.2.6 Ефективність тестування 41
3.3 Fastlane *41*
3.3.1 Створення lane для тестування 41
3.3.2 Інтеграція з XCUITests 42
3.3.3 Автоматизація процесу 43
3.4 Висновки до розділу *43*
ВИСНОВКИ **45**
ДЖЕРЕЛА **46**

АНОТАЦІЯ

В даній роботі розглянуто вдосконалену систему для автоматизованого тестування адаптивності інтерфейсу iOS/iPadOS додатків. Проводиться дослідження існуючих механізмів використання XCUITest, CucumberSwift, fastlane та Figma. Розглядається поєднання цих інструментів для створення єдиного, універсального, який буде спрощувати процес тестування інтерфейсу додатків. Система має на меті спростити впровадження автоматичного тестування інтерфейсу, порівняння обмежень елементів на екрані та ідентифікація результатів. Новостворений засіб тестування може також послугувати покращенню вже існуючих додатків, за рахунок того, що макет еталонного дизайну створюється з максимальною точністю, а активні обмеження елементів інтерфейсу напряму порівнюються відповідно з тими що у додатку.

ВСТУП

В сучасному світі, де мобільні додатки стають неодмінною частиною повсякденного життя, актуальність розробки та тестування інтерфейсів для iOS/iPadOS додатків набуває все більшої вагомості. Однак, зростаюча складність додатків та різноманітність пристроїв, на яких вони запускаються, ставлять перед розробниками нові виклики у сфері тестування.

Обрана тема дослідження, щодо вдосконаленої системи для автоматизованого тестування адаптивності інтерфейсу iOS/iPadOS додатків, обгрунтовується актуальністю не лише у контексті зростання популярності мобільних додатків, а й у зв'язку з необхідністю забезпечення якості в їхньому функціонуванні. Розробка інструментів для автоматичного тестування графічного інтерфейсу стає ключовою для забезпечення продуктивності та ефективності процесу розробки.

Предметом дослідження є система автоматизованого тестування адаптивності інтерфейсу iOS/iPadOS додатків, що базується на поєднанні інструментів XCUITest, CucumberSwift, fastlane та веб-додатку Figma. Метою дослідження є розробка універсального інструменту, що спростить процес автоматичного тестування мобільних додатків, зокрема, забезпечить ефективну перевірку адаптивності їхніх інтерфейсів.

Огляд джерел, пов'язаних із темою, спрямовує на те, що не існує інструменту, який би був схожий на наш та вирішував би проблеми описані вище. Попереднє судження вказує на необхідність розробки нових підходів до тестування мобільних додатків. Методологічна основа дослідження ґрунтується на аналізі існуючих методів та інструментів тестування, а також на розробці нового інструменту, що враховує сучасні вимоги та тенденції у розробці мобільних додатків.

Роботу поділено на три розділи, кожний з яких висвітлює кожний інструмент окремо відповідно до кожного етапу розробки обраного підходу до розробки загальної системи тестування.

Перший розділ розглядає теоретичні основи інструментів, які використовуються у дослідженні. Починаючи з Figma, проводиться огляд можливостей та переваг його використання для дизайну та прототипування мобільних додатків. Далі аналізується вплив Figma на процес розробки мобільних додатків. Другий пункт розділу присвячено CucumberSwift, де проводиться огляд методів тестування мобільних додатків та розглядається роль CucumberSwift у реалізації автоматизованих тестів. Нарешті, розглядаються переваги використання fastlane у процесі розробки мобільних додатків, а також проводиться огляд основних функцій та інструментів цього інструменту.

У другому розділі розглядається вибір та обґрунтування методів тестування з використанням кожного з інструментів. Починаючи з Figma, розглядається візуалізація тестових кейсів, колаборативний процес та інші аспекти. Далі досліджується BDD підхід з використанням CucumberSwift, а також переваги автоматизованого тестування інтерфейсу користувача. У третьому пункті розділу розглядаються переваги fastlane у процесі тестування, зокрема, автоматизація тестування, інтеграція з іншими інструментами та інші аспекти.

У третьому розділі надається опис імплементації кожного з інструментів та представлені результати їх використання. Розглядаються кроки завантаження та обробки даних з Figma, порівняння макетів з реальним інтерфейсом та ефективність процесу тестування. Далі розглядається імплементація CucumberSwift, включаючи налаштування середовища, написання тестових сценаріїв та їх ефективність. У третьому пункті розділу описується імплементація fastlane, зокрема, створення lane для тестування, інтеграція з XCUITests та ефективність тестування.

РОЗДІЛ 1. ПРОЕКТУВАННЯ ТА ТЕСТУВАННЯ ГРФІЧНОГО ІНТЕРФЕЙСУ

1.1 Figma

1.1.1 Огляд можливостей та переваг користування Figma для дизайну і прототипування.

Дизайн мобільних додатків є складним та багатогранним процесом, що вимагає уваги до дрібниць та точності в кожному етапі. Від відображення інтерфейсу до взаємодії користувача, кожний елемент повинен бути продуманим та оптимізованим для досягнення найкращого враження користувача.

Одним із ключових аспектів успішної розробки додатків є врахування різних розмірів та складностей, які можуть виникнути в процесі розробки. Figma, як інструмент для дизайну та прототипування, допомагає розробникам ефективно вирішувати ці завдання.

Завдяки широкому спектру налаштувань та можливостей, Figma дозволяє розробникам будувати додатки різних розмірів та складностей з урахуванням індивідуальних потреб кожного проекту. Від мінімалістичних інтерфейсів до складних систем з великою кількістю функцій, Figma надає необхідні інструменти для того, щоб кожен елемент додатку був продуманим та оптимізованим для досягнення найкращого враження користувача.

Крім того, Figma має свій власний API, який відкриває безліч можливостей для отримання детальної інформації про кожен елемент дизайну. За допомогою API розробники можуть отримувати доступ до різноманітних даних, таких як розміри, кольори, шрифти та багато іншого, що дозволяє забезпечити ще більшу точність та деталізацію в процесі розробки додатку.

Таким чином, Figma відкриває перед розробниками широкі можливості для творення дизайну мобільних додатків будь-якої складності та розміру, забезпечуючи не лише ефективність та точність, але й гнучкість та індивідуальний підхід до кожного проекту.

Широкий спектр можливостей Figma відкриває перед розробниками, спрощуючи та прискорюючи процес розробки мобільних додатків. Основні переваги використання Figma полягають у можливості створення високоякісних макетів. Figma надає широкі можливості для створення дизайну мобільних додатків з високою деталізацією та якістю. Розробники можуть легко створювати ефективні макети інтерфейсу користувача. Друга головна перевага – це спільна робота над проектами. Розробники можуть легко ділитися своїми макетами з колегами, робити спільні правки та коментувати різні аспекти дизайну, що дозволяє збільшити швидкість та ефективність розробки. Також існують всі можливості для швидкої зміни та ітерації дизайну. Figma дозволяє швидко змінювати та вдосконалювати дизайн мобільних додатків. Завдяки відсутності необхідності завантажувати чи оновлювати програмне забезпечення, розробники можуть миттєво вносити зміни та перевіряти їх ефективність. Та останнє – це інтерактивність та прототипування. Figma дозволяє створювати інтерактивні прототипи мобільних додатків, що допомагає розробникам краще розуміти потенційні можливості та функціонал додатку перед його реалізацією. Це дозволяє виявити та усунути можливі проблеми ще на етапі проектування.

Загалом, використання Figma значно спрощує та прискорює процес дизайну мобільних додатків, забезпечуючи розробникам необхідні інструменти для створення високоякісного та ефективного інтерфейсу користувача. Це дозволяє зосередитися на створенні унікального та інноваційного досвіду для користувачів та забезпечує успішне впровадження мобільних додатків на ринок[5].

1.1.2 Аналіз впливу Figma на процес розробки мобільних додатків.

Аналіз впливу Figma на процес розробки мобільних додатків відображає значні переваги та позитивні зміни, які цей інструмент приніс у галузі дизайну та розробки програмного забезпечення. Розглянемо приклад існуючого популярного додатку та його впливу після переходу на використання Figma.

Ще однією важливою перевагою використання Figma була здатність проводити зручне та ефективне тестування ітерацій дизайну. Команда могла швидко створювати прототипи, вносити зміни та тестувати їх з реальними користувачами без зайвих зусиль. Це дозволило виявляти та усувати проблеми ще на ранніх етапах розробки, що значно економило час та ресурси.

Крім цього, використання Figma API дозволило автоматизувати деякі процеси та інтегрувати роботу з макетами у інші інструменти, що ще більше оптимізувало процес розробки.

Dev Mode в Figma надає все необхідне для навігації по файлам дизайну та перетворення дизайну в код. За допомогою Dev Mode, дизайнери та розробники можуть знаходитися на одній хвилі, переконуючись, що важливі деталі не загубляються в процесі передачі.

Розробники використовують Dev Mode для:

- Доступу до розширених можливостей інспекції та більшої кількості мов кодування.
- Впевненості, що вони працюють з останніми специфікаціями, легко порівнюючи версії рамок.
- Швидкого перегляду дизайнів, які готові до розробки, з використанням статусів та анотацій.

- Оптимізації робочих процесів за допомогою інтеграцій, спрямованих на розробників, таких як JIRA, Storybook та GitHub.
- Дослідження всіх варіантів у наборі компонентів без редагування файлу.
- Пов'язання дизайнів з завданнями від менеджерів, документацією та кодовими компонентами.

Значною перевагою використання Figma є можливість брати зображення у потрібних форматах прямо з дизайну. Це дозволяє розробникам швидко і легко отримувати необхідні ресурси для використання у своїх проектах, що сприяє збереженню часу та оптимізації робочих процесів.

Варто зауважити, що для оптимального співробітництва між дизайнерами та розробниками, дизайнерам важливо старатися забезпечити, щоб дизайн додатку у Figma був повним, детальним та зрозумілим для розробників. Це можна досягти шляхом правильної організації ієрархії елементів та компонентів, а також дотримання всіх правил та стандартів дизайну.

Кілька ключових аспектів, на які дизайнерам варто звернути увагу для забезпечення зрозумілості та легкості розуміння дизайну розробниками – це повна і детальна специфікація, чітка ієрархія view та відповідність стандартам і практикам.

Розглянемо аспекти детальніше. Перший означає, що дизайнери повинні намагатися враховувати всі можливі деталі та вимоги проекту у своєму дизайні у Figma. Це включає в себе правильне відображення всіх елементів і стилів, а також додаткові вказівки щодо інтерактивності та анімації. Другий - структура дизайну повинна бути організованою та логічною, з належною ієрархією view та груп елементів. Це допоможе розробникам легше розібратися у структурі дизайну та швидше здійснювати розробку. Та останній ключовий аспект має на увазі, що дизайнерам слід дотримуватися встановлених стандартів та найкращих практик

дизайну. Це спростить розуміння дизайну розробниками та допоможе уникнути непорозумінь та неполадок під час реалізації.

З вищезазначених відомостей Figma є центральним зв'язком для всіх учасників команди, що дозволяє швидко узгоджувати правки у дизайні та реалізовувати новий функціонал. Це сприяє покращенню комунікації та зменшенню часу, потрібного для розробки додатків, що є критичним у сучасному швидкому темпі розвитку технологій.

Отже, аналіз впливу Figma на процес розробки мобільних додатків підтверджує значний ріст продуктивності та якості кінцевого продукту. Впровадження цього інструменту дозволило розробникам працювати більш ефективно, збільшуючи швидкість та якість розробки.

1.2 CucumberSwift

1.2.1 Огляд методів тестування мобільних додатків.

У сучасному світі тестування мобільних додатків є необхідною складовою процесу розробки, оскільки воно допомагає забезпечити якість продукту та задовольнення користувачів. Одним із ключових аспектів тестування є UI тести, які спрямовані на перевірку коректності роботи інтерфейсу додатка. Нижче будуть описані загальні методи тестування мобільних додатків та їх вплив на процес розробки.

Розглянемо перший підхід - ручне тестування: це традиційний метод, при якому тести виконуються вручну людиною без використання автоматизованих інструментів. Ручне тестування може бути часом затратним та витратним, але воно дозволяє виявляти проблеми, які можуть бути важко автоматизувати.

Другий приклад – це автоматизоване тестування: воно використовує скрипти та інструменти для автоматизації процесу тестування. Це включає в себе різні види тестів, такі як функціональні тести, регресійні тести та тести на витривалість. У контексті мобільних додатків, автоматизоване тестування дозволяє швидше виявляти та виправляти помилки, зберігаючи якість продукту на високому рівні.

Третій випадок, які є предметом нашого дослідження – це UI тести: вони перевіряють, чи відображається інтерфейс користувача додатка вірно та чи працюють всі його елементи на різних пристроях та платформах. Вони відтворюють дії користувача та перевіряють, чи відбуваються очікувані зміни в інтерфейсі. UI тести допомагають уникнути неполадок, пов'язаних з відображенням інтерфейсу, і забезпечують користувачам приємний досвід взаємодії з додатком[7].

Використання Cucumber, зокрема, забезпечує написання тестів у формі зрозумілій для людини, що сприяє кращій співпраці у команді. Автоматизація тестування інтерфейсу користувача з CucumberSwift дозволяє забезпечити коректну роботу додатків на різних пристроях та платформах, що є ключовим аспектом у розробці додатків. Подальше впровадження автоматизованого тестування, зокрема UI тестів, несе потенційно високу якість додатків після проходження відповідних тестових випадків[10].

1.2.2 Роль CucumberSwift у реалізації автоматизованих тестів

Зважаючи на зростаючу складність розробки програмного забезпечення та потребу у постійній комунікації між учасниками проекту, стає відомо, що прозорість та зрозуміння вимог до продукту вкрай важливі для успішної реалізації. Це саме той момент, коли виникає необхідність в методології, яка не

лише дозволить узгоджувати вимоги між різними учасниками команди, але й сприятиме створенню чіткої та зрозумілої документації.

Behavior-driven development (BDD) - це гнучкий підхід до розробки, в якому три практики застосовуються до кожної історії/елементу backlog: виявлення, формулювання та автоматизація. Про BDD написано багато, і є хороші вступні статті, але потрібно звернути увагу, що ці практики повинні застосовуватися в правильному порядку. Виявлення, потім формулювання, потім автоматизація.[1]

В рамках BDD, автоматизація означає створення тестових сценаріїв перед написанням коду, який буде перевірятися цими тестами. Аналогічно до Test-Driven Development (TDD), невдалий автоматизований тест викликає необхідність удосконалення вихідного коду. Цей підхід має два важливі наслідки. Перший свідчить, що кожен тест повинен бути перевірений на невдачу, коли він написаний, і перевірений на проходження, коли реалізований правильний виробничий код. Не слід покладатися на результати тесту, достовірність якого не була підтверджена через проведення належної перевірки. Другий наслідок – це те, що код автоматизації повинен бути написаний або людьми, які будуть писати виробничий код, або кимось, хто дуже тісно з ними співпрацює. Одним із ключових аспектів BDD є використання природної мови для опису функціональності продукту, що робить процес комунікації між розробниками, тестувальниками та бізнес-аналітиками більш ефективним та зрозумілим[9].

Одним з інструментів, які надають можливості втілення BDD в життя є Cucumber: інструмент командного рядка, який зчитує наші специфікації з текстових файлів простою мовою, що називаються функціями. Він перевіряє ці функції на наявність сценаріїв для тестування і запускає ці сценарії на нашій системі. Кожен сценарій представляє собою список кроків, які виконує Cucumber. Щоб Cucumber міг виконати ці сценарії, вони повинні відповідати певним вимогам. Файли функцій повинні відповідати базовим правилам синтаксису,

відомим як gherkin. Крім функцій, ми також надаємо Cucumber набір визначень кроків, які перетворюють читабельну мову кроків у код Ruby для виконання відповідних дій. У зрілому тестовому наборі визначень кроку буде лише кілька рядків Ruby, які викликають допоміжний код нашого додатку, зазвичай бібліотеки автоматизації, наприклад, Capybara, для взаємодії з системою. [2]

CucumberSwift - це фреймворк для автоматизованого тестування, що ґрунтується на концепціях Behavior-Driven Development (BDD). Цей інструмент дозволяє розробникам писати тестові сценарії в природній мові та виконувати їх для перевірки функціональності програмного забезпечення. Однією з ключових відмінностей CucumberSwift від оригінального Cucumber є його спрощена інтеграція з мовою програмування Swift, що робить його ідеальним вибором для розробки мобільних додатків для платформ iOS та macOS.

Порівняно з аналогічними інструментами для інших мов програмування, CucumberSwift надає більшу зрозумілість та легкість в інтеграції з мовою Swift. Він дозволяє розробникам писати тестові сценарії безпосередньо на Swift, що робить код тестів більш читабельним та зрозумілим. Крім того, CucumberSwift має багато вбудованих функцій та можливостей, які спрощують процес написання, виконання та аналізу тестів, забезпечуючи ефективну автоматизацію тестування мобільних додатків.

Далі буде розглянуте питання цього підрозділу, ролі та висновки про обраний інструмент. Однією з головних ролей CucumberSwift є забезпечення спільного розуміння вимог до програмного забезпечення між всіма учасниками команди, включаючи розробників, тестувальників та менеджерів продукту. Завдяки цьому, команди можуть визначити очікувані результати та функціональні можливості додатку на ранніх стадіях розробки.

Крім того, CucumberSwift допомагає автоматизувати тестування програмного забезпечення, що значно зменшує час та зусилля, необхідні для ручного

тестування. Він надає можливість автоматично виконувати тестові сценарії на різних конфігураціях та платформах, що сприяє виявленню помилок та проблем у ранніх етапах розробки.

Однією з важливих переваг CucumberSwift є зручний та зрозумілий синтаксис для написання тестових сценаріїв, що дозволяє навіть не-технічним учасникам команди активно брати участь у процесі тестування. Такий підхід сприяє покращенню комунікації в команді та забезпечує більшу відкритість та взаєморозуміння між всіма сторонами проекту.

Загалом, роль CucumberSwift у реалізації автоматизованих тестів полягає в тому, щоб забезпечити ефективну комунікацію в команді, автоматизувати процес тестування та сприяти швидкому виявленню та виправленню помилок у програмному забезпеченні.

1.3 Fastlane

1.3.1 Переваги використання Fastlane у процесі розробки мобільних додатків.

Перед тим як досліджувати переваги використання fastlane, будуть розглянуті загальні відомості про цей засіб автоматизації. Fastlane - це інструмент для розробників iOS та Android для автоматизації таких нудних завдань, як створення скріншотів, робота з профілями забезпечення та випуск нашого додатку.[3] І цей інструмент на всі 100% впорається с поставленими задачами. Ось детальний опис переваг fastlane та те, як вони можуть допомогти та де їх втілити. Перший і головний аспект – це зберігти години кожного разу, коли ми випускаємо нову версію в магазин або сервіс для бета-тестування: Ця перевага означає, що ми економимо значний час, який раніше витрачали на ручне виконання рутинних завдань під час випуску нових версій нашого додатка. Використання fastlane дозволяє автоматизувати ці процеси та зменшити ризик помилок. По-друге, він

інтегрується з усіма існуючими інструментами та сервісами (більше 400 інтеграцій): fastlane підтримує безліч інтеграцій з різними сервісами та інструментами, що дозволяє нам легко включити його в наш поточний стек розробки. Також, важливо, що це 100% відкрите джерело під ліцензією MIT, що дозволяє бути повністю відкритим джерелом забезпечує гнучкість та можливість самостійно налаштовувати та розширювати fastlane відповідно до нашого бізнесу та потреб розробки. Ще це простий помічник для швидкого початку роботи за кілька хвилин. Fastlane надає легкий та швидкий процес налаштування, який дозволяє вам швидко розпочати використання інструменту без зайвих зусиль та заплутаності. Додамо, що він працює на нашому комп'ютері, це наш додаток і наші дані. Fastlane запускається на нашому комп'ютері, що забезпечує нам повний контроль над нашими даними та додатками. Дуже зручно, що інструмент інтегрується з усіма основними системами неперервної інтеграції. Fastlane легко інтегрується з основними системами, такими як Jenkins, CircleCI, GitLab CI тощо, що дозволяє автоматизувати весь цикл розробки. Дуже важливо для нашої роботи, що Fastlane підтримує iOS, Mac та Android додатки. Він працює як з мобільними, так і з настільними додатками, що робить його універсальним інструментом для всіх платформ. Розширення та налаштування fastlane під свої потреби: fastlane дозволяє розширити та налаштувати його функціональність відповідно до нашого проекту та потреб, що робить його дуже гнучким та адаптованим до нашого процесу розробки. Також можна забути про команди, просто fastlane. Засіб надає зручний та інтуїтивний інтерфейс, який дозволяє нам виконувати складні завдання лише однією командою, що зекономить наш час та зусилля. Є можливість запускати з будь-якого комп'ютера, включаючи сервер CI: fastlane дозволяє нам легко випускати наші додатки з будь-якого комп'ютера, включаючи сервери неперервної інтеграції, що забезпечує гнучкість та зручність в управлінні процесом розробки.

Після того, як ми розглянули переваги використання Fastlane, можемо побачити, що цей інструмент дійсно вирішує низку проблем, головна з яких

витрата великої кількості часу на збір Завдяки його автоматизації та інтеграції з існуючими інструментами та сервісами, Fastlane допомагає ефективно випускати нові версії додатків, зберігаючи час та зусилля розробників. Цей інструмент є відкритим джерелом, що надає йому гнучкість та можливість налаштування під конкретні потреби проекту. Його простий інтерфейс та можливість виконання складних завдань за допомогою однієї команди забезпечує зручність в користуванні, тим самим полегшуючи роботу розробників.

Fastlane також дозволяє розширити свою функціональність за допомогою розширень та налаштувань, що робить його вельми гнучким та адаптованим до потреб розробки. Забезпечення підтримки для різних платформ, таких як iOS, Mac та Android, робить його універсальним інструментом для всіх типів додатків. Крім того, можливість легкого випуску додатків з будь-якого комп'ютера, включаючи сервери неперервної інтеграції, підвищує ефективність та зручність процесу розробки.

Отож, Fastlane є потужним інструментом, який допомагає розробникам мобільних додатків зосередитися на створенні високоякісного програмного забезпечення, мінімізуючи витрати часу та зусиль на рутинні завдання.

1.3.2 Огляд основних функцій та інструментів Fastlane

Fastlane має низку корисних функцій та інструментів нижче буде подано список основних з них.

1. **Match**: Цей інструмент дозволяє автоматично керувати та оновлювати сертифікати та профілі забезпечення для розробки iOS і Android, забезпечуючи безпроблемну роботу з цими складними складовими процесу розробки.

2. **Scan**: Scan автоматизує процес запуску тестів на різних пристроях та платформах, дозволяючи швидко та ефективно виявляти помилки та проблеми у нашому додатку.
3. **Gym**: Gym використовується для збирання та підготовки нашого додатку для випуску, дозволяючи автоматизувати процеси компіляції та пакування програмного забезпечення.
4. **Deliver**: Deliver спрощує процес випуску нашого додатку в App Store та Google Play Store, дозволяючи завантажувати нові версії програми та керувати метаданими додатків з командного рядка.
5. **Spaceship**: Spaceship надає доступ до Apple Developer Center з командного рядка, що дозволяє автоматизувати процеси, пов'язані з управлінням програмними ресурсами та іншими аспектами розробки для платформи iOS.
6. **Frameit**: Frameit дозволяє автоматично додавати кадри до скріншотів нашого додатку, щоб вони виглядали стильно та професійно, що полегшує підготовку матеріалів для промоції.
7. **Snapshot**: Snapshot дозволяє автоматично створювати скріншоти нашого додатку на різних пристроях та платформах для використання в App Store та Google Play Store.

Ці інструменти допомагають автоматизувати ключові аспекти процесу розробки та випуску мобільних додатків, що забезпечує ефективність та швидкість у розробці програмного забезпечення.

1.4 Висновки

Figma виявляється потужним інструментом для дизайну та прототипування, який забезпечує зручність у співпраці між дизайнерами та розробниками, спрощуючи процес створення та редагування макетів та покращуючи командну роботу.

Використання Figma має значний вплив на процес розробки мобільних додатків, зменшуючи час на створення дизайну, підвищуючи зрозумілість вимог та полегшуючи співпрацю між учасниками проекту.

CucumberSwift є ефективним інструментом для автоматизованого тестування мобільних додатків, особливо в контексті Behaviour-Driven Development (BDD). Він спрощує процес створення та виконання тестів, допомагаючи виявляти помилки та вдосконалювати функціональність додатків.

Fastlane забезпечує значні переваги у процесі розробки мобільних додатків, автоматизуючи та спрощуючи завдання, такі як збірка, тестування та реліз програмного забезпечення. Його широкий набір інструментів дозволяє ефективно керувати різними аспектами розробки та забезпечує швидкий та надійний випуск додатків до користувачів.

Загалом, використання цих інструментів сприяє покращенню продуктивності, якості та швидкості розробки мобільних додатків, що дозволяє розробникам та дизайнерам зосередитися на створенні значущих та інноваційних продуктів.

РОЗДІЛ 2. АВТОМАТИЗАЦІЯ ТЕСТУВАННЯ UI

2.1 Figma

2.1.1 Візуалізація тестових кейсів

На першому етапі розробки мобільних додатків, Figma відіграє ключову роль у візуалізації тестових кейсів. Дизайн у Figma є еталонним зразком з еталонними значеннями, який можна використовувати для орієнтації під час тестування. Це дозволяє тестувальникам докладно ознайомитися з кожною деталлю дизайну, від найменших елементів інтерфейсу до загальної композиції.

Дизайн у Figma є не лише ілюстрацією того, яким має бути кінцевий продукт, але й основою для порівняння з фактичним інтерфейсом додатку. Візуалізація тестових кейсів у Figma дозволяє команді виявити потенційні проблеми та недоліки в дизайні ще на ранніх етапах розробки. Це дозволяє зменшити кількість помилок та забезпечити високу якість кінцевого продукту.

Використання макету у Figma як еталону дозволяє тестувальникам отримати доступ до дизайну у найдрібніших деталях. Це дозволяє їм точно аналізувати та перевіряти кожен аспект інтерфейсу додатку на відповідність вимогам та очікуванням користувачів. Такий підхід сприяє виявленню потенційних проблем ще на ранніх етапах розробки та дозволяє забезпечити високу якість кінцевого продукту.

2.1.2 Колаборативний процес

Figma є не лише інструментом для дизайну, але й платформою, яка створює сприятливі умови для колаборації між всіма учасниками розробницької команди, включаючи дизайнерів, тестувальників та розробників.

Завдяки Figma, всі члени команди можуть працювати над проектом у реальному часі, спільно вносячи зміни, роблячи коментарі та взаємодіючи один з

одним. Це полегшує процес співпраці, сприяє швидкій зворотній зв'язку та дозволяє уникнути затримок у розробці через недопорозуміння або неузгодженості.

Зокрема, Figma дозволяє розробникам отримувати доступ до дизайну та прототипів безпосередньо в середовищі, з якого вони працюють. Це спрощує розуміння вимог до дизайну та можливості взаємодії між дизайнерами та розробниками під час процесу розробки.

2.1.3 Тестування на ранніх етапах розробки

На ранніх етапах розробки, коли доступний лише один модуль або функціональність додатку, використання Figma може бути особливо корисним для тестування інтерфейсу. Оскільки весь дизайн та інтерфейс додатку можуть бути розроблені та протестовані в Figma ще до початку фактичної розробки програми, це дозволяє проводити тестування на ранніх стадіях.

Наприклад, якщо у фірмі готовий лише один модуль або окрема частина додатку, дизайн цієї частини може бути розроблений та протестований за допомогою Figma. Завдяки цьому інструменту тестувальники можуть вже на ранніх етапах отримати доступ до дизайну, перевірити його на відповідність вимогам та функціональності, та внести свої пропозиції щодо вдосконалення. В нашому випадку цей підхід допоможе створювати тест-кейси при появі перших екранів, що скорочує час створення тестів в подальшому.

Такий підхід дозволяє виявити можливі проблеми та недоліки в дизайні на ранніх стадіях, що дозволяє зменшити кількість виправлень під час пізніших етапів розробки та збільшує шанси на успішне впровадження продукту. Крім того, це також зменшує ризик змін у вимогах та затримок у розробці, оскільки всі сторони вже можуть погодитися щодо вигляду та функціональності додатку на ранніх етапах.

2.1.4 Ефективне порівняння з дійсним інтерфейсом

Ефективне порівняння макету з реальним інтерфейсом є важливою складовою використання Figma. Благодаря тому, що макети у Figma побудовані на базі інструменту Figma Auto Layout, який повністю відтворює концепцію Auto Layout у UIKit, можливо створювати інтерфейси, що відображають ті самі обмеження, що й реальний дизайн у iOS додатках.

Цей механізм побудований на принципі створення обмежень (constraints), що дозволяють визначати розташування та позицію елементів інтерфейсу в межах view. Такий підхід дозволяє точно відтворити структуру інтерфейсу згідно з заданими параметрами, забезпечуючи відповідність між макетом та реальним інтерфейсом.

Це значно спрощує процес порівняння та аналізу, оскільки дозволяє розробникам та тестувальникам оцінити, наскільки точно макет відображає фактичний інтерфейс додатка.

2.2 CucumberSwift

2.2.1 BDD (Behavior-Driven Development) підхід

BDD, або Behavior-Driven Development (розробка з урахуванням поведінки), використовується як основний підхід у CucumberSwift з численних причин.

По-перше, BDD забезпечує зрозуміння та спільне уявлення про те, як повинен функціонувати продукт, яке спрощує комунікацію між різними учасниками проекту, такими як розробники, тестувальники та бізнес-аналітики. Це дозволяє уникнути недорозумінь та непорозумінь щодо очікувань стосовно функціоналу продукту.

По-друге, BDD дозволяє описати поведінку продукту з точки зору користувача, а не технічні деталі. Це сприяє покращенню зрозуміння вимог до продукту, особливо для тестувальників, та створює можливість для більш ефективного валідації цих вимог.

Крім того, BDD впроваджує принцип "тести, що описують поведінку", де тести написані у формі специфікацій, а не просто набору тестових випадків. Це дозволяє створити зрозумілі та чіткі тести, які можуть бути використані як засіб комунікації між учасниками проекту та як документація до програмного коду.

Отже, використання BDD в CucumberSwift дозволяє створювати автоматизовані тести, які не лише валідують роботу програми, але й є зрозумілими та придатними для спільної роботи між усіма членами команди розробки.

2.2.2 Легкість читання та зміни тестів

Легкість читання та зміни тестів у CucumberSwift визначається його зрозумілим та природним синтаксисом, який використовується для написання тестових сценаріїв.

Наприклад, в CucumberSwift тестові сценарії описуються у вигляді сценаріїв узгодження (scenarios)(рис. 2.1). Кожен сценарій може містити кілька кроків (steps), які описують послідовні дії, які потрібно виконати для перевірки певного функціоналу.

Ось приклад простого сценарію в CucumberSwift:

```
Feature: Validate Figma layout in the iOS app

Scenario: Verify layout of the buttons stack view matches Figma design
  Given the Figma document is fetched
  When the app is launched
  Then the buttons stack view layout should match the Figma design
```

Рис 2.1. CucumberSwift Scenario приклад

У цьому прикладі кожен крок (Given, When, Then) відповідає певній дії, яку потрібно виконати. Наприклад, Given описує початковий стан системи, When - дії, які ми виконуємо, і Then - очікуваний результат.

Такий синтаксис дозволяє команді легко читати та розуміти тестові сценарії, а також змінювати їх у відповідності до змін у вимогах або функціоналу програми. Це полегшує процес тестування та сприяє швидкому виявленню та виправленню помилок.

2.2.3 Автоматизація тестування інтерфейсу користувача

Автоматизація тестування інтерфейсу користувача за допомогою CucumberSwift може бути реалізована наступним чином. По-перше треба провести визначення тестових сценаріїв та створити Feature файл. Перш за все, тестувальники визначають тестові сценарії, які описують поведінку системи з точки зору користувача. Ці сценарії зазвичай і записуються у вигляді "Feature" файлів, які містять описи різних функціональних можливостей програми. Другий етап – це написання кроків (Step Definitions). Після визначення тестових сценаріїв, тестувальники визначають реалізацію кожного кроку (Given, When, Then) у вигляді методів, які будуть виконувати відповідні дії під час тестування. Ці методи називаються "step definitions". Далі підготовка середовища, для виконання тестів необхідно підготувати тестове середовище, включаючи запуск додатку або веб-сайту, який буде тестуватися, та забезпечення необхідних даних для тестування.

Потім у нас йде запуск тестів. Після написання тестових сценаріїв та відповідних кроків, тести можна запустити за допомогою спеціальних інструментів, таких як XCTest або XCUI Test для iOS додатків[8]. Ці інструменти дозволяють автоматично виконувати тестові сценарії та перевіряти, чи відповідає реальна реакція програми очікуваному результату. У нашому випадку цим буде займатись інший засіб, який опишемо та розглянемо у наступному підрозділі Fastlane.

Після завершення тестів необхідно проаналізувати їх результати, щоб виявити та виправити будь-які виявлені помилки або невідповідності. Cucumber точно скаже на якому кроці у нас виявився хибний результат і позначить його червоним.

Таким чином, CucumberSwift дозволяє автоматизувати тестування інтерфейсу користувача шляхом опису тестових сценаріїв у зрозумілій для всіх сторінній мові та реалізації кроків для кожного з них. Це полегшує розуміння та підтримку тестів, а також забезпечує стабільність та надійність програми.

2.2.4 Інтеграція з іншими інструментами

Інтеграція CucumberSwift з іншими інструментами відбувається через Xcode та fastlane. Оскільки fastlane відповідає за запуск тестів, він може автоматично запускати процес тестування за допомогою CucumberSwift. Це означає, що як XCTest тести, так і тестові сценарії з Cucumber можуть бути запущені за допомогою fastlane.

Щодо інтеграції з Xcode, CucumberSwift легко інтегрується з середовищем розробки Xcode, що дозволяє зручно використовувати його в процесі розробки мобільних додатків. Бібліотека CucumberSwift надає зручний інтерфейс для інтеграції з Xcode, що дозволяє розробникам швидко і ефективно виконувати автоматизоване тестування інтерфейсу користувача без зайвих зусиль.

2.3 Fastlane

2.3.1 Автоматизація тестування

Автоматизація тестування з fastlane відіграє ключову роль у процесі розробки мобільних додатків. Завдяки fastlane розробники можуть автоматизувати виконання тестів, що дозволяє їм швидше виявляти помилки в під час роботи додатку.

Ще однією важливою перевагою автоматизації тестування за допомогою fastlane є можливість швидко реагувати на зміни в коді. Після кожного оновлення програмного забезпечення fastlane автоматично запускає тестові набори, що дозволяє розробникам оперативно виявляти будь-які нові помилки або проблеми.

Крім того, fastlane надає можливість інтеграції з іншими інструментами, такими як системи керування версіями, сервіси для неперервної інтеграції та постачання (CI/CD), що спрощує процес розробки та тестування мобільних додатків. В результаті, розробники отримують швидкий та ефективний механізм автоматизації тестування, що допомагає їм зосередитися на самому процесі розробки і забезпечити високу якість продукту[11].

2.3.2 Інтеграція з Xcode та XCTest

Інтеграція fastlane з Xcode та XCTest надає розробникам потужний засіб для автоматизації тестування їх мобільних додатків. Далі буде детальніше описано про цей процес. Розробники можуть створити конфігураційний файл Fastfile у своєму проекті. Цей файл містить скрипти, які визначають, які кроки повинен виконати fastlane для автоматизації тестування та випуску додатку. У Fastfile можуть бути визначені скрипти для запуску тестів XCTest, які відповідають за запуск конкретних тестових наборів або всіх тестів у проекті. Розробники можуть налаштувати ці скрипти залежно від своїх потреб.

Fastlane може викликати команди Xcode через командний рядок, щоб виконати тестування. Він використовує Xcodebuild, інструмент командного рядка Xcode, для компіляції та запуску тестів. Розробники можуть передавати параметри до своїх тестових скриптів через fastlane. Це дозволяє їм налаштовувати тестування для різних середовищ або конфігурацій.

Fastlane може легко інтегруватися з різними сервісами для неперервної інтеграції, такими як Jenkins, CircleCI, або GitLab CI. Це дозволяє розробникам

створювати різні автоматизовані робочі процеси, які включають в себе тестування, збірку та випуск додатків.

Загалом, інтеграція fastlane з Xcode та XCTest дозволяє розробникам легко і ефективно автоматизувати процес тестування своїх мобільних додатків, що сприяє покращенню якості програмного забезпечення та прискоренню процесу розробки.

2.3.3 Послідовність тестування

Послідовність тестування визначається за допомогою файлів функцій Cucumber. У цих файлах сценарії тестування описуються у вигляді конкретних кроків, які повинні бути виконані для перевірки різних аспектів додатка. Порядок цих сценаріїв визначає порядок виконання тестів. Коли CucumberSwift запускає тести, він читає ці файли та виконує кожен сценарій у порядку, в якому вони зустрічаються у файлі функцій.

Це означає, що розробники можуть контролювати послідовність тестування, структуруючи сценарії тестування у файлі функцій відповідно до їх потреб. Наприклад, вони можуть спочатку перевіряти базові функції додатка, а потім переходити до більш складних або важливих аспектів.

Таким чином, CucumberSwift допомагає регулювати послідовність тестування, забезпечуючи гнучкий та контрольований підхід до автоматизованого тестування мобільних додатків.

2.3.4 Простота налаштування та використання

Простота налаштування та використання Fastlane для загальних випадків є однією з його ключових переваг. Fastlane пропонує інтуїтивний та швидкий процес налаштування, який дозволяє розробникам швидко розпочати використання інструменту без зайвих зусиль та заплутаності.

Fastlane надає широкий набір готових команд та акцій, які можна використовувати для автоматизації різних аспектів розробки та випуску додатків. Ці команди включають в себе такі операції, як збірка, тестування, генерація скріншотів, підписання додатку, випуск в магазин та багато іншого.

У цілому, Fastlane робить процес автоматизації випуску додатків простим та ефективним, забезпечуючи розробникам можливість швидко налаштувати та використовувати інструмент для автоматизації різних завдань у своєму проекті.

2.3.5 Інтеграція з іншими інструментами

Інтеграція з іншими інструментами у Fastlane дозволяє розробникам поєднувати його з різними сервісами та інструментами, щоб створювати повний інструментарій для автоматизації розробки та випуску додатків. Як ми зазначали вище, Fastlane інтегрується з такими CI/CD системами, як Jenkins, CircleCI, GitLab CI та іншими. Це дозволяє налаштовувати автоматичні процеси збірки, тестування та релізу додатків при кожному зміні в коді. Наприклад, після кожного коміту у репозиторії, CI/CD система може автоматично викликати Fastlane, який запустить необхідні тести, упакує та підготує додаток для релізу. Такий підхід дозволяє забезпечити стабільність та якість додатків, а також значно зменшити час, необхідний для розгортання змін. Наш lane з запуском UI тестів не виключення, його легко можна включити в цей ланцюг процесів.

2.3.6 Використання Fastlane для запуску процесу тестування

Зараз опишемо причини з котрих ми обрали підхід запуску тестів через Fastlane. Їх існує декілька. Перша - fastlane дозволяє автоматизувати рутинні завдання, в нашому випадку це систематичний запуск тестів, що значно підвищує ефективність розробки та тестування. Використовуючи fastlane, ми можемо легко налаштувати та запускати тести без необхідності вручну виконувати кожен крок, що економить час і зусилля команди. По-друге, fastlane забезпечує інтеграцію з

іншими інструментами, такими як Xcode та XCTest, що дозволяє створювати потужні та гнучкі сценарії автоматизації. Це включає запуск тестів у середовищі CI/CD, що забезпечує безперервну інтеграцію та доставку. Такий підхід дозволяє автоматично виконувати тести при кожному зміненні коду, забезпечуючи швидке виявлення та виправлення помилок. Використання fastlane також забезпечує узгодженість і стабільність процесів. Завдяки можливості створення lane для різних задач, ми можемо стандартизувати процеси тестування та забезпечити, що вони виконуються однаково на всіх етапах розробки та на різних машинах. Це допомагає уникнути помилок, пов'язаних з людським фактором, та забезпечує високу якість продукту.

2.4 Висновки до розділу

У цьому розділі було розглянуто вибір та обґрунтування методів тестування мобільних додатків. На початку розділу було проведено огляд можливостей та переваг використання Figma для дизайну і прототипування, включаючи візуалізацію тестових кейсів, колаборативний процес, тестування на ранніх етапах розробки та ефективне порівняння з дійсним інтерфейсом. Далі було розглянуто методи тестування мобільних додатків з використанням CucumberSwift, зосереджуючись на його ролі у реалізації автоматизованих тестів та перевагах, які він надає, таких як BDD підхід, легкість читання та зміни тестів, автоматизація тестування інтерфейсу користувача, а також інтеграція з іншими інструментами. Завершивши розділ, було розглянуто fastlane, де обговорювалась автоматизація тестування, інтеграція з Xcode та XCTest, послідовність тестування, простота налаштування та використання, а також інтеграція з іншими інструментами.

РОЗДІЛ 3. ОПИС ІМПЛЕМЕНТАЦІЇ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Figma

3.1.1 Завантаження та обробка даних з Figma

Цей підрозділ містить опис розробленого інструменту з способу завантаження вихідних даних з Figma. Figma API надає можливість завантажувати дані з макетів Figma безпосередньо у програми або сервіси для подальшої обробки. Це робиться шляхом взаємодії з відповідними ендпоінтами, які надають різноманітні дані, такі як структура макетів, графічні елементи, тексти, шари та інші компоненти. Розглянемо приклад ендпоінту для отримання даних. “https://api.figma.com/v1/files/(fileID)” , де fileID – це id нашого файлу Figma з дизайном додатку. Також потрібен header у вигляді token, який можна згенерувати на у налаштуваннях свого профілю. Цей header потрібно вставити в конфігурацію запиту для коректної роботи.

Після отримання доступу до Figma API та відповідної автентифікації можна здійснювати запити до API для завантаження даних з конкретних макетів. Наприклад, можна отримати структуру макету у вигляді JSON-об'єкта, який містить інформацію про шари, групи, елементи та їх властивості. Це саме те, що нам потрібно для реалізації інструменту. Кожен елемент ієрархіїї має своє ім’я, яке в подальшому ми будемо використовувати його як ідентифікатор для парних елементів(тобто з дизайну Figma та з iOS/iPadOS додатку). Найважливіші параметри, які ми будемо застосовувати для тестування адаптивності інтерфейсу – це відступи(paddings) елементів, тобто відступи з усіх сторін дочірніх елементів від батьківських. Такий підхід означає, що точність тестування буде залежати від правильної конфігурації тестів для окремих елементів дизайну[6].

Отже, завантажені дані можуть бути оброблені подальше для різноманітних цілей, таких як аналіз макету, автоматичне створення коду, генерація документації або проведення тестування відповідності макетів.

Цей процес дає змогу розробникам та тестувальникам ефективно використовувати дані з макетів Figma в своїх проектах безпосередньо з інтегрованих середовищ програмування або систем управління версіями.

3.1.2 Порівняння макетів з реальним інтерфейсом

Використання даних з Figma API для порівняння окремих елементів інтерфейсу є найефективнішим джерелом з яким можна порівнювати дизайн на відповідність. На прикладі елемента UIStackView можемо провести наступні кроки. По-перше, отримання даних з Figma API, за допомогою API ми можемо отримати всю необхідну інформацію про макет, включаючи ідентифікатори елементів, їх розміри, положення, типи та інші властивості. По-друге, ідентифікація цільових елементів, тобто ми можемо знаходити цільові елементи за допомогою їх ідентифікаторів або інших унікальних властивостей, таких як accessibilityIdentifier. Наприклад, для UIStackView ми можемо знайти елементи з відповідним ідентифікатором. По-третє, порівняння характеристик елементів: Після отримання даних про цільові елементи з Figma та їх властивостей, ми можемо порівняти їх з фактичними характеристиками елементів у додатку. Наприклад, ми можемо порівняти розміри, положення, відстань до супервью, колір, шрифт та інші параметри. Розглянемо приклад на стандартних відступах праворуч та ліворуч по 20 одиниць для елементу на дизайні. Елемент, який ми тестуємо, має мати ці ж самі відступи, які вираховуються за допомогою наших створених допоміжних функцій. Вони просто вираховуються актуальний відступ з кожної з сторін. Більш примітивні способи порівняння також присутні – це визначення чи правильно вказана ширина або висота, якщо це має потребу. Ми самі можемо визначати, які саме обмеження нам потрібно перевіряти. Щоб точно

не помилитись на Figma є точні прототипи обмежень, які мають бути присутні у додатку для певних елементів дизайну. Потім йде етап виявлення розбіжностей, якщо під час порівняння виявляються розбіжності між характеристиками макету та реалізацією в додатку, система може згенерувати повідомлення про помилку або сповістити процес розробки про потребу у внесенні змін. Про це поговоримо у наступних підрозділах про Cucumber та Fastlane. Останній пункт - це автоматизація процесу. Щоб забезпечити ефективне використання цього методу, можна автоматизувати процес порівняння та виявлення розбіжностей. Це дозволить швидко виявляти та виправляти помилки у відповідності між макетом та реалізацією.

3.1.3 Створення додатку на основі макетів

Створення додатку на основі макетів передбачає суворе відповідання реалізації дизайну задокументованим макетам та використання інструменту Figma Auto Layout. Це означає, що дизайнер має створити макет, який відображає усі необхідні елементи та їх характеристики відповідно до вимог функціональності та дизайну додатку. Важливо, щоб макет був стабільним і коректним, адже від нього буде залежати подальша імплементація додатку.

Під час створення додатку на основі макетів, розробники повинні точно відтворити всі елементи та їх взаємодію відповідно до макета. Це включає розміри, положення, кольори, шрифти та інші важливі характеристики. Для забезпечення відповідності макету під час імплементації можна використовувати дані з Figma file, щоб перевіряти та порівнювати окремі елементи та їх властивості. Відповідне ставлення до норм дизайне буде потім напряму перевірятись за цим же дизайном, тому його дотримання обов’язкове.

Отже, створення додатку на основі макетів передбачає відповідність дизайну усім задокументованим вимогам та створення досконалого макету з усіма

необхідними елементами, який можна буде використовувати для тестування за допомогою даних з Figma file.

Для прикладу розглянемо Stack View(рис. 3.1), який містить у собі дві кнопки, які мають певні обмеження.

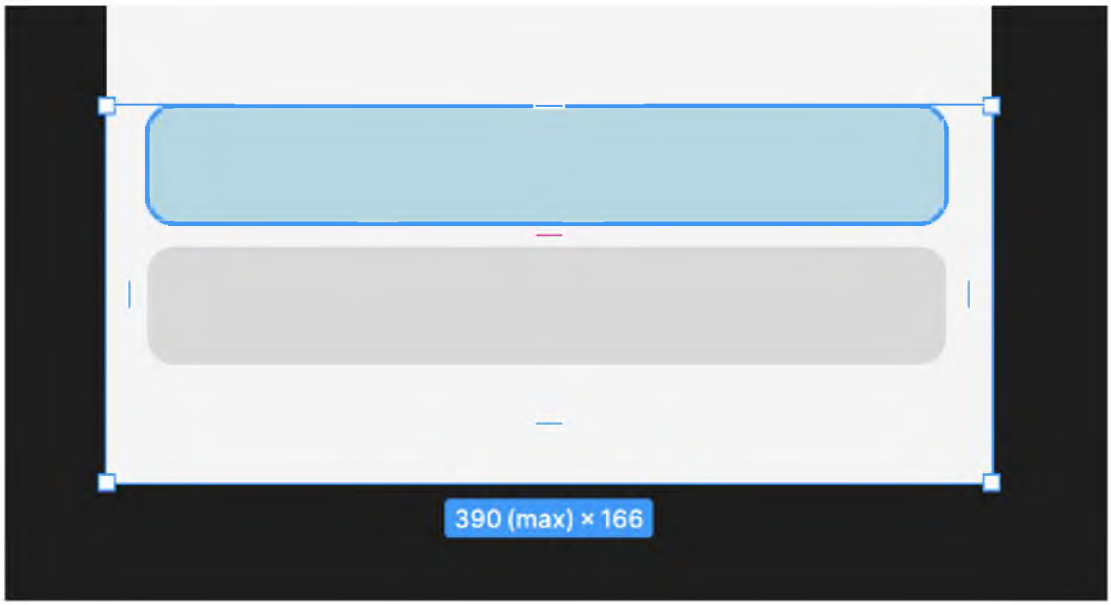

Рис 3.1. Приклад вигляду елементів дизайну у Figma

Відповідні позначки у вигляді рисок з першого погляду кидаються в око, це і є наші встановленні обмеження у дизайні, які ми будемо перевіряти пізніше під час проходження тестів.

3.1.4 Тестування відповідності макетів

Для тестування відповідності макетів можна використовувати функції порівняння елементів, подібні до наведеної. Наприклад, функція assertLeftPadding() приймає об'єкт FigmaFrame, який містить інформацію про відступи макету, та XCUIElement, який є елементом, знайденим у додатку за допомогою XCTest. Функція порівнює лівий відступ елемента зі значенням, яке вказано у макеті.

Важливо зазначити, що напряму порівнювати constraints за допомогою XCUIElement не можна, оскільки об'єкти XCUIElement не мають прив'язаних

constraints. Тому для порівняння відступів були створені допоміжні функції, які вираховують відступи між елементами та їх superview. Ці функції додані як розширення до фреймворку CucumberSwift, щоб можна було використовувати їх у тестах. Такий підхід дозволяє ефективно порівнювати відповідність макетів у додатку. Наведемо приклад такої функції:

```
func assertLeftPadding(_ figmaFrame: FigmaFrame,
                       matches appElement: XCUIElement,
                       inSuperview superview: XCUIElement,
                       file: StaticString = #file, line: UInt = #line
) {
    guard let paddingLeft = figmaFrame.paddingLeft else {
        XCTFail("Figma frame doesn't have left padding", file: file, line: line)
        return
    }

    XCTAssertEqual(appElement.frame.minX - superview.frame.minX, paddingLeft, file: file, line: line)
}
```

Лістинг 3.1. Приклад пошуку constraint ліворуч відносно superview

3.1.5 Ефективність процесу тестування

Тестування відповідності макетів за допомогою даних з Figma є дуже ефективним процесом. Цей підхід дозволяє точно визначити відхилення від еталону інтерфейсу, оскільки макети в Figma зазвичай мають детальні описи кожного елемента. Порівняно з іншими методами, такими як порівняння скріншотів, де важко визначити конкретні відхилення, тестування за допомогою макетів є більш ефективним і надійним.

Основна перевага полягає в тому, що ми можемо створити детальні тестові сценарії, які перевіряють відповідність кожного елемента макету з його реальним представленням у додатку.

3.2 CucumberSwift

3.2.1 Налаштування середовища для розробки тестів

Щоб налаштувати середовище для розробки тестів з використанням CucumberSwift, спочатку потрібно встановити сам CucumberSwift у свій проект. Це можна зробити через CocoaPods або Swift Package Manager.

Після інсталяції CucumberSwift слід створити тестові файли, що міститимуть тестові сценарії у форматі Gherkin. Ці файли будуть містити опис кроків для тестування різних функціональностей додатку.

Далі необхідно налаштувати тестове середовище у своєму проекті. Це може включати конфігураційні файли, налаштування тестових цілей та інші параметри, які дозволять виконувати тести.

Після цього слід визначити самі тестові сценарії, які будуть перевірятися за допомогою CucumberSwift. Це включає в себе опис кроків для тестування різних аспектів функціональностей додатку.

І на останок, для зручності тестування можна налаштувати інтеграцію CucumberSwift з Xcode, щоб запускати тести прямо з інтерфейсу розробника.

Такий підхід допоможе налаштувати середовище для ефективної розробки тестів з використанням CucumberSwift та розпочати процес написання та виконання тестів для додатку.

3.2.2 Написання тестових сценаріїв

Написання тестових сценаріїв відбувається у папці Features, яка є стандартною практикою для фреймворку CucumberSwift. У цій папці описуються тестові сценарії за допомогою мови Gherkin, яка є структурованою мовою для опису поведінки програмного забезпечення.

Кожен тестовий сценарій починається з ключового слова, яке визначає тип сценарію, такий як Scenario або Scenario Outline. Після цього слідують кроки, які

описують послідовність дій, що повинні відбуватися під час виконання тесту. Кожен крок може бути асоційований з певним кроком в коді тестів.

Наприклад, ось приклад тестового сценарію для перевірки введення користувачем даних у форму реєстрації:

```
Feature: Registration

Scenario: User fills out registration form
  Given the user is on the registration screen
  When the user enters their name "John" and email "john@example.com"
  And the user sets a password "adminpass"
  And the user taps the submit button
  Then the user should see a confirmation message
```

Текст 3.1. Приклад сценарію для тестування появи повідомлення на екрані

У цьому прикладі описано тестовий сценарій, де користувач заповнює форму реєстрації. Кожен крок цього сценарію буде відповідати певному кроку в коді тестування. Після відображення повідомлення вікно повідомлення зазвичай має бути однакової висоти та ширини у всіх розмірах екранів, тому можна одразу за допомогою наших засобів перевірити відповідність дизайну.

3.2.3 Створення самостійних тестів

Описані тестові сценарії демонструють структуру та синтаксис використання бібліотеки CucumberSwift для написання автоматизованих тестів інтерфейсу користувача.

У першому тестовому кроці (Given), за допомогою замикання, виконується запит до Figma API для отримання документа з макетом. Відбувається перевірка наявності цього документа, і якщо він успішно отриманий, він зберігається у змінній figmaDocument.

У другому кроці (When) відбувається запуск додатку. Використовується стандартний метод launch() об'єкту XCUIApplication.

У третьому кроці (Then) проводиться порівняння розміщення кнопок у макеті Figma з їх розміщенням у реальному додатку. Для цього використовуються попередньо написані допоміжні функції для порівняння параметрів розміщення (висоти, лівого та правого відступів тощо) елементів інтерфейсу.

У разі, якщо тест не пройде, викликаються методи XCTFail(), які вказують на помилку та надають додаткову інформацію про її причину.

Такий підхід дозволяє автоматизувати тестування відповідності макетів дизайну та реального інтерфейсу додатку з використанням CucumberSwift.

3.2.4 Інтеграція системи порівняння layouts

Для інтеграції системи порівняння layouts з бібліотекою CucumberSwift, було розроблено набір допоміжних функцій, які розширюють можливості CucumberSwift та дозволяють ефективно порівнювати параметри розміщення елементів інтерфейсу між макетом в Figma та реальним додатком. Давайте розглянемо основні з цих функцій:

1. **assertFigmaDocumentFetched(_:)**: Перевіряє, чи був успішно отриманий документ з макетом з Figma. Якщо документ не був отриманий, викликає помилку.
2. **assertFigmaFrameFound(_:)**: Перевіряє, чи був успішно знайдений рамка (frame) в макеті Figma. Якщо рамка не була знайдена, викликає помилку.
3. **assertFrameWidth(_:matches:)**: Перевіряє відповідність ширини рамки між макетом в Figma та реальним елементом інтерфейсу. Якщо ширини не співпадають, викликає помилку.
4. **assertFigmaFrameHeight(_:matches:)**: Перевіряє відповідність висоти рамки між макетом в Figma та реальним елементом інтерфейсу, з урахуванням відступів. Якщо висоти не співпадають, викликає помилку.

5. **assertTopPadding(_:matches:inSuperview:)**: Перевіряє відповідність верхнього відступу рамки між макетом в Figma та реальним елементом інтерфейсу. Якщо відступи не співпадають, викликає помилку.
6. **assertRightPadding(_:matches:inSuperview:)**: Перевіряє відповідність правого відступу рамки між макетом в Figma та реальним елементом інтерфейсу. Якщо відступи не співпадають, викликає помилку.
7. **assertLeftPadding(_:matches:inSuperview:)**: Перевіряє відповідність лівого відступу рамки між макетом в Figma та реальним елементом інтерфейсу. Якщо відступи не співпадають, викликає помилку.
8. **assertBottomPadding(_:matches:inSuperview:)**: Перевіряє відповідність нижнього відступу рамки між макетом в Figma та реальним елементом інтерфейсу. Якщо відступи не співпадають, викликає помилку.

Ці функції дозволяють автоматизувати порівняння параметрів розміщення елементів між макетом в Figma та реальним додатком, що значно спрощує та прискорює процес тестування відповідності макетів.

3.2.5 Виконання тестових сценаріїв

Виконання тестових сценаріїв здійснюється за допомогою засобів Xcode або fastlane для забезпечення більшої гнучкості та можливості інтеграції в процес CI/CD.

Використання засобів Xcode для виконання тестових сценаріїв є стандартним підходом, оскільки Xcode надає зручний інтерфейс для запуску тестів, перегляду результатів і відлагодження.

Однак, для досягнення більшої автоматизації тестування та інтеграції його в процес CI/CD, ми можемо скористатися інструментом fastlane. Fastlane дозволяє автоматизувати різні аспекти процесу розробки, включаючи запуск тестів, збірку, підписування і розгортання додатків.

Інтеграція з fastlane дозволяє використовувати різноманітні плагіни та скрипти для керування процесом тестування. Це може включати автоматичний

запуск тестів при кожному зміні коду, генерацію звітів про прогони тестів, інтеграцію з сервісами збірки та навіть розгортання нових версій додатків після успішного завершення тестів. Основне звичайно саме залучення нашого інструменту до стандартного пакету функцій Fastlane.

Такий підхід дозволяє забезпечити надійність та стабільність додатку шляхом автоматичного виконання тестів на кожному етапі розробки та забезпечення відповідності функціональності вимогам.

3.2.6 Ефективність тестування

Розглянемо ефективність тестування з перспективи CucumberSwift. Використання цього фреймворку для написання тестових сценаріїв, описаних за допомогою BDD, забезпечує зрозумілі вимоги до функціональності програмного забезпечення. Завдяки BDD підходу та опису тестових сценаріїв у вигляді природної мови, розробники можуть більш повно тестувати функціонал додатку, покриваючи різноманітні варіанти використання.

3.3 Fastlane

3.3.1 Створення lane для тестування

Для конфігурації Fastlane та створення lane вам потрібно встановити Fastlane на нашому комп'ютері та налаштувати його для нашого проекту. Нижче будуть описані кроки, які потрібно виконати для досягнення поставленої мети.

Головне середовище для створення lanes є Fastfile. У ньому треба визначити lane для виконання певних завдань, для нас це тестування.

```
platform :ios do
  desc "Testing layout"
  lane :run_layout_tests do
  run_tests(project: "UITestingCucumber.xcodeproj",
            devices: ["iPhone 15 Pro"],
            scheme: "UITestingCucumber")
  end
end
```

Лістинг 3.2. Приклад lane

У цьому прикладі run_layout_tests - це назва нашого lane, і він викликає run_test, щоб запустити наші тести інтерфейсу користувача. Lane може бути конфігурований згідно з поставленими цілями тестування. Наприклад, визначити специфічні моделі екранів для тестування layout.

Налаштування параметрів та змінних: Ми можемо додати параметри та змінні до нашого Fastfile, щоб забезпечити більшу гнучкість і перевикористання. Наприклад, ми можемо використовувати змінні для збереження шляхів до нашого додатку або конфігураційних даних. Виклик lane з командного рядка або інтегрувати його у наш процес CI/CD.

3.3.2 Інтеграція з XCUITests

Отримавши конфігурацію Xcode проекту, Fastlane може інтегруватися з XCUITests через виклик команди run_tests, як було описано раніше. Ця інтеграція дозволяє автоматизувати запуск тестів, а також їх відстеження та аналіз результатів. Таким чином, Fastlane стає потужним інструментом для автоматизації тестування мобільних додатків, забезпечуючи швидке та ефективне виконання тестів у процесі розробки.

Коли тести запускаються через Fastlane у командному рядку, результати виконання тестів виводяться у консоль. Це може бути інформація про успішне чи невдале виконання окремих тестів, загальний статус тестів, а також будь-які

повідомлення про помилки чи відхилення. Такий формат дозволяє розробникам швидко отримувати звіти про стан тестування без необхідності відкривати інші інтерфейси. Також варто зауважити, що через те що ми поєднали Fastlane з CucumberSwift, то ми будемо одразу бачити тест-кейс, який пройшов не успішно і одразу зрозуміти проблемне місце.

3.3.3 Автоматизація процесу

Автоматизація процесу в даному випадку полягає в тому, що за допомогою Fastlane можна налаштувати процес запуску тестів автоматично, без необхідності в ручному виконанні кожного тесту. Крім того, результати тестування можуть автоматично записуватись та аналізуватись, що спрощує виявлення помилок та відхилень у коді.

3.4 Висновки до розділу

У цьому розділі ми детально розглянули процес тестування мобільних додатків, використовуючи різноманітні інструменти та методи.

Починаючи з використання Figma для візуалізації тестових кейсів та порівняння макетів з реальним інтерфейсом, ми перейшли до використання CucumberSwift для реалізації тестування на основі BDD підходу та автоматизації тестів інтерфейсу користувача. Ми також детально розглянули інтеграцію цих інструментів з Xcode та іншими інструментами розробки, такими як fastlane.

Реалізація fastlane дозволяє нам автоматизувати виконання тестів, використовуючи XCTest та інші технології, інтегруючи їх в CI/CD процес розробки. Це забезпечує ефективне та надійне тестування додатків, спрощуючи процес виявлення помилок та покращення якості продукту.

Загалом, використання цих інструментів та методів сприяє покращенню ефективності та якості процесу розробки мобільних додатків, дозволяючи швидше виявляти та виправляти помилки, забезпечуючи кращий користувацький досвід.

ВИСНОВКИ

Створений інструмент для тестування інтерфейсу iOS додатків є значним кроком у покращенні якості та ефективності процесу розробки програмного забезпечення. Цей інструмент розроблено з урахуванням різних важливих аспектів для забезпечення його зручного та ефективного використання.

Основна перевага цього інструменту полягає в тому, що він дозволяє точно порівнювати макети і реальний інтерфейс, виявляючи навіть найменші недоліки і розбіжності. Крім того, методологія BDD в поєднанні з Cucumber полегшує написання та інтерпретацію тестових скриптів, особливо для невеликих команд розробників. Ви можете автоматизувати тестування і включити його в поточний конвеєр CI/CD шляхом інтеграції з Fastlane.

В результаті розробки цього інструменту з'явився потужний і корисний інструмент для оцінки користувацьких інтерфейсів iOS-додатків, який можна застосовувати для підвищення якості і надійності програмного забезпечення.

ДЖЕРЕЛА

1. Behaviour-Driven Development [Електронний ресурс] – Режим доступу до ресурсу: https://cucumber.io/docs/bdd/.
2. Matt W. The Cucumber Book. Behaviour-Driven Development for Testers and Developer / W. Matt, H. Aslak., 2012. – 307 с.
3. fastlane docs [Електронний ресурс] / team fastlane – Режим доступу до ресурсу: https://docs.fastlane.tools
4. fastlane docs iOS Setup [Електронний ресурс] / fastlane – Режим доступу до ресурсу: https://docs.fastlane.tools/getting-started/ios/setup/.
5. Figma docs [Електронний ресурс] / figma – Режим доступу до ресурсу: https://help.figma.com/hc/en-us.
6. Figma REST API [Електронний ресурс] / figma – Режим доступу до ресурсу: https://www.figma.com/developers/api.
7. Penn J. Test iOS Apps with UI Automation: Bug Hunting Made Easy / Jonathan Penn., 2013. – 201 с.
8. Tsadok A. Pro iOS Testing: XCTest Framework for UI and Unit Testing / Avi Tsadok. – Tel Mond, Israel: Apress, 2020. – 295 с.
9. Ferguson J. S. BDD in action. Behavior-Driven Development for the whole software lifecycle / John Smart Ferguson. – Shelter Island: MANNING, 2015. – 345 с.
10. Axelrod A. Complete Guide to Test Automation Techniques, Practices, and Patterns for Building and Maintaining Effective Software Projects / Arnon Axelrod. – New York: Apress, 2018. – 515 с.
11. Humble J. Continuous Delivery: Reliable Software Releases Build through, Test, and Deployment Automation / J. Humble, D. Farley. – Boston: Pearson Education, 2011. – 445 с.