КОМАНДА ЗВУКА

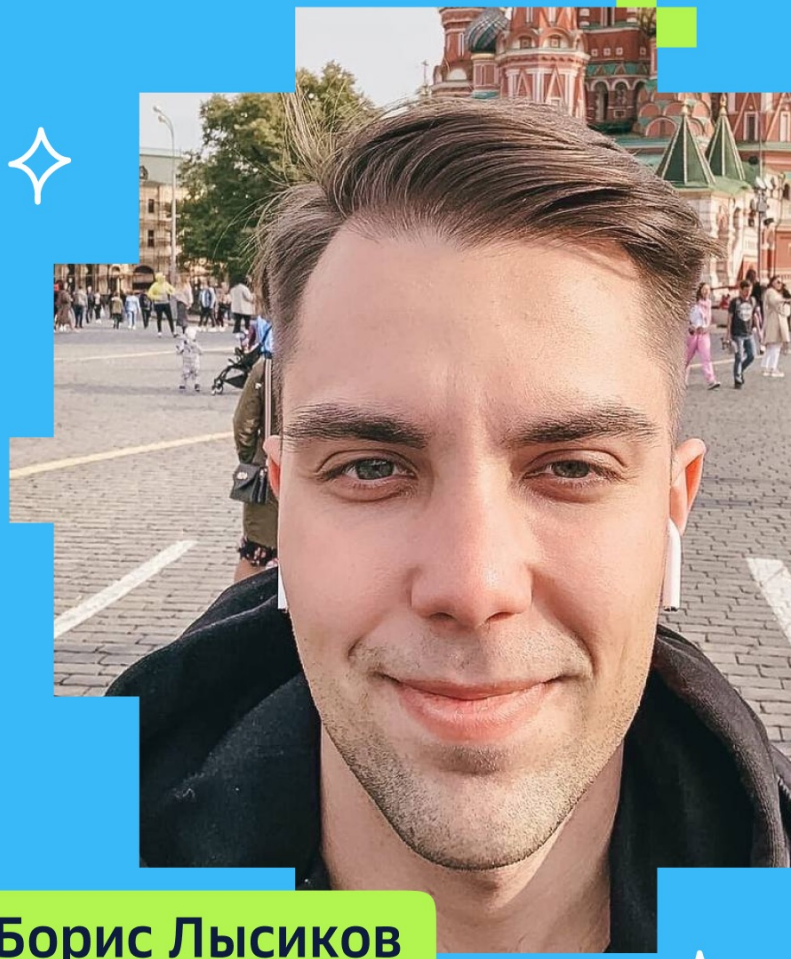
Как ускорить прохождения iOS UI-тестов

с 4 часов до 30 минут

КОМАНДА ЗВУКА

Пара слов о себе

- в QA 5+ лет
- писал автотесты на платформы: iOS, Backend, Web, Android
- автор тг-канала об iOS-автоматизации
- автор цикла статей об iOS-автоматизации



Борис Лысиков

Руководитель отдела автоматизированного тестирования

Немного внутренней кухни

- 1 AQA iOS, 6 manual QA только начинающих писать ui-тесты;
- **Стэк:** Swift + XCTest + TinkoffMockStraping + Marathon + Allure Testops
- **Железо:** 2 mac mini M2 Pro, 16 GB ram
- **Количество UI-тестов:** 378
- **Время прохождения:** 4 часа 12 минут

Проблема

378 ui-тестов занимало 4 часа 12 минут с учетом сборки проекта

Как можно сократить

1. Анализ текущих автотестов

Анализ текущих автотестов

Этапы:

1. Определяем критерии того, что не стоит проверять ui-тестами

Анализ текущих автотестов

Этапы:

1. Определяем критерии того, что не стоит проверять ui-тестами
2. Фиксируем это в документации

Анализ текущих автотестов

Этапы:

1. Определяем критерии того, что не стоит проверять ui-тестами
2. Фиксируем это в документации
3. Проводим общую встречу с Manual QA

Анализ текущих автотестов

Этапы:

1. Определяем критерии того, что не стоит проверять ui-тестами
2. Фиксируем это в документации
3. Проводим общую встречу с Manual QA
4. Удаляем ненужные автотесты и переводим их на ручной флоу

Что не стоит покрывать ui-тестами

1. Кейсы, которые требует человеческого взаимодействия

Что не стоит покрывать ui-тестами

1. Кейсы, которые требует человеческого взаимодействия
2. Кейсы по фиче, которая находится в эксперименте

Что не стоит покрывать ui-тестами

1. Кейсы, которые требует человеческого взаимодействия
2. Кейсы по фиче, которая находится в эксперименте
3. Кейсы, в которых есть анимация на странице

Как можно сократить

1. Анализ текущих автотестов
2. Убираем компиляцию проекта

Убираем компиляцию проекта

1. Формируем файлы для запуска

Убираем компиляцию проекта

1. Формируем файлы для запуска
2. Сохраняем их

Убираем компиляцию проекта

1. Формируем файлы для запуска
2. Сохраняем их
3. Запускаем тесты по ним

Какие файлы нужны?

Раннер	Файлы
xcodebuild	.xctestrun || derived data

Какие файлы нужны?

Раннер	Файлы
xcodebuild	.xctestrun || derived data
fastlane	.xctestrun || derived data

Какие файлы нужны?

Раннер	Файлы
xcodebuild	.xctestrun || derived data
fastlane	.xctestrun || derived data
marathon	.app && .xctest || derived data

Формируем нужные файлы

xcodebuild

```
xcodebuild build-for-testing [-workspace <your_workspace_name>]
                             [-project <your_project_name>]
                             -scheme <your_scheme_name>
                             -destination <destination-specifier>
```

Формируем нужные файлы

fastlane

```
run_tests(
    derived_data_path: "my_folder",
    build_for_testing: true
)
```

Запускаем тесты

xcodebuild

```
xcodebuild test-without-building -xctestrun <your_xctestrun_name>.xctestrun
                                 -destination <destination-specifier>
                                 [-only-testing:<test-identifier>]
                                 [-skip-testing:<test-identifier>]
```

Запускаем тесты

fastlane

```
run_tests(
    derived_data_path: "my_folder",
    test_without_building: true
)
```

Запускаем тесты

fastlane

```
run_tests(
   xctestrun: "/path/to/mytests.xctestrun"
)
```

Как можно сократить

1. **Анализ текущих автотестов**
2. **Убираем компиляцию проекта**
3. **Оптимизация флоу ui-тестов**
    a. Поднимаем экраны

Способы поднятие экранов

1. Deep-links

Способы поднятие экранов

1. Deep-links
2. Использование ArgumentHandler для подмены стартового View

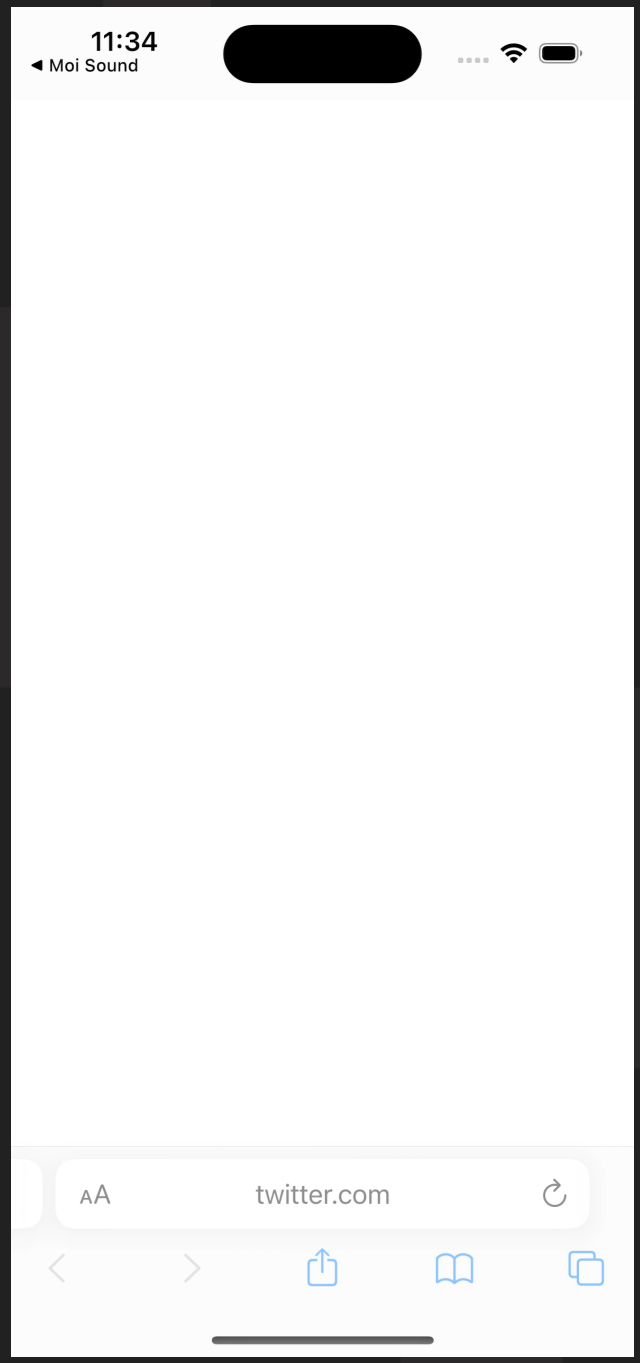
Deep-links

Xcode версии < 14.3

```
let safari = XCUIApplication(bundleIdentifier: "com.apple.mobilesafari")
let url = safari.textFields.firstMatch
let alertOpenButton = safari.otherElements["SFDialogView"].buttons.element(boundBy: 1)

safari.activate()
url.waitForExistence(timeout: 2)
url.tap()
url.waitForExistence(timeout: 2)
url.typeText("yourApp://deeplink")
safari.keyboards.buttons["go"].waitForExistence(timeout: 2)
safari.keyboards.buttons["go"].tap()
alertOpenButton.waitForExistence(timeout: 2)
alertOpenButton.tap()
```

Время прохождения: 9 секунд



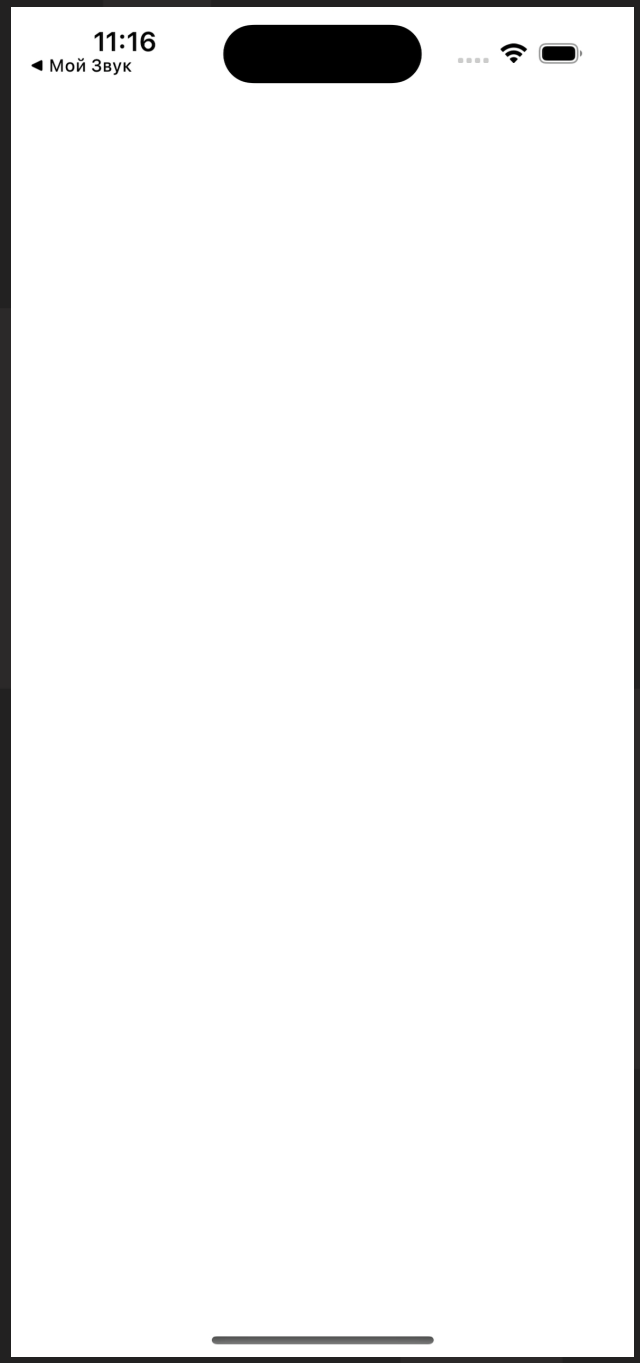
Deep-links

Xcode версии > 14.3

```
let safari = XCUIApplication(bundleIdentifier: "com.apple.mobilesafari")
let alertOpenButton = safari.otherElements["SFDialogView"].buttons.element(boundBy: 1)

safari.open(URL(string: "yourApp://deeplink")!)
alertOpenButton.waitForExistence(timeout: 2)
alertOpenButton.tap()
```

Время прохождения: 4 секунды



Плюсы и минусы Deep-links подхода

Плюсы

- Простая реализация и поддержка
- Проверка корректности работы диплинков

Минусы

- Все экраны должны поддерживать диплинки

Использование ArgumentHandler для подмены стартового View

1. Прокидывание аргументов или переменных окружения из таргета с тестами

Использование ArgumentHandler для подмены стартового View

1. Прокидывание аргументов или переменных окружения из таргета с тестами
2. Создание ArgumentHandler для подмены стартового View

Использование ArgumentHandler для подмены стартового View

1. Прокидывание аргументов или переменных окружения из таргета с тестами
2. Создание ArgumentHandler для подмены стартового View
3. Использование ArgumentHandler в жизненном цикле

Прокидывание аргументов

UIKit & SwiftUI

```
override func setUp() {
    XCUIApplication().launchArguments.append("Screen")
    XCUIApplication().launchEnvironment["Screen"] = "DetailsScreen"
}
```

Создание ArgumentHandler

UIKit

```
struct ArgumentHandler {
    static func handleView() -> UIViewController {
        let arguments = ProcessInfo.processInfo.arguments

        if arguments.contains("DetailView") {
            return DetailViewController()
        }
        return MainViewController()
    }
}
```

Создание ArgumentHandler

SwiftUI

```
struct ArgumentHandler: View {
    var body: some View {
        let arguments = ProcessInfo.processInfo.arguments

        if arguments.contains("DetailView") {
            DetailView()
        } else {
            ContentView()
        }
    }
}
```

Использование ArgumentHandler в жизненном цикле

UIKit

```
class AppDelegate: UIResponder, UIApplicationDelegate {
    var window: UIWindow?
    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
        [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

        let window = UIWindow()
        #if DEBUG
        window.rootViewController = ArgumentHandler.handleView()
        #else
        window.rootViewController = MainViewController()
        #endif
        window.makeKeyAndVisible()

        self.window = window

        return true
    }
}
```

Использование ArgumentHandler в жизненном цикле

SwiftUI

```
@main
struct PublicArtApp: App {
    var body: some Scene {
        WindowGroup {
            #if DEBUG
                ArgumentHandler()
            #else
                ContentView()
            #endif
        }
    }
}
```

Плюсы и минусы использования ArgumentHandler для подмены стартового View

Плюсы

- Экран поднимается моментально

Минусы

- Нужна помощь разработчиков, либо хороший уровень погружения в проект и разработку
- Больше кода

Как можно сократить

1. **Анализ текущих автотестов**
2. **Убираем компиляцию проекта**
3. **Оптимизация флоу ui-тестов**
    a. Поднимаем экраны
    b. Используем моки

Виды мок-серверов

1. local (Swifter, TinkoffMockStraping)

Виды мок-серверов

1. local (Swifter, TinkoffMockStraping)
2. standalone (Wiremock)

Виды мок-серверов

1. local (Swifter, TinkoffMockStraping)
2. standalone (Wiremock)
3. interceptor (SBTUITestTunnel)

Плюсы и минусы мок-сервера

Плюсы

- Скорость прохождения тестов
- Создание контролируемых сценариев
- Улучшение стабильности тестов

Минусы

- Риск недостоверности
- Сложность поддержания моков

Как можно сократить

1. **Анализ текущих автотестов**
2. **Убираем компиляцию проекта**
3. **Оптимизация флоу ui-тестов**
    a. Поднимаем экраны
    b. Используем моки
    c. Делаем ограничение на время прохождения автотеста

Ограничение на время прохождения ui-теста

В коде executeTimeAllowance = 60

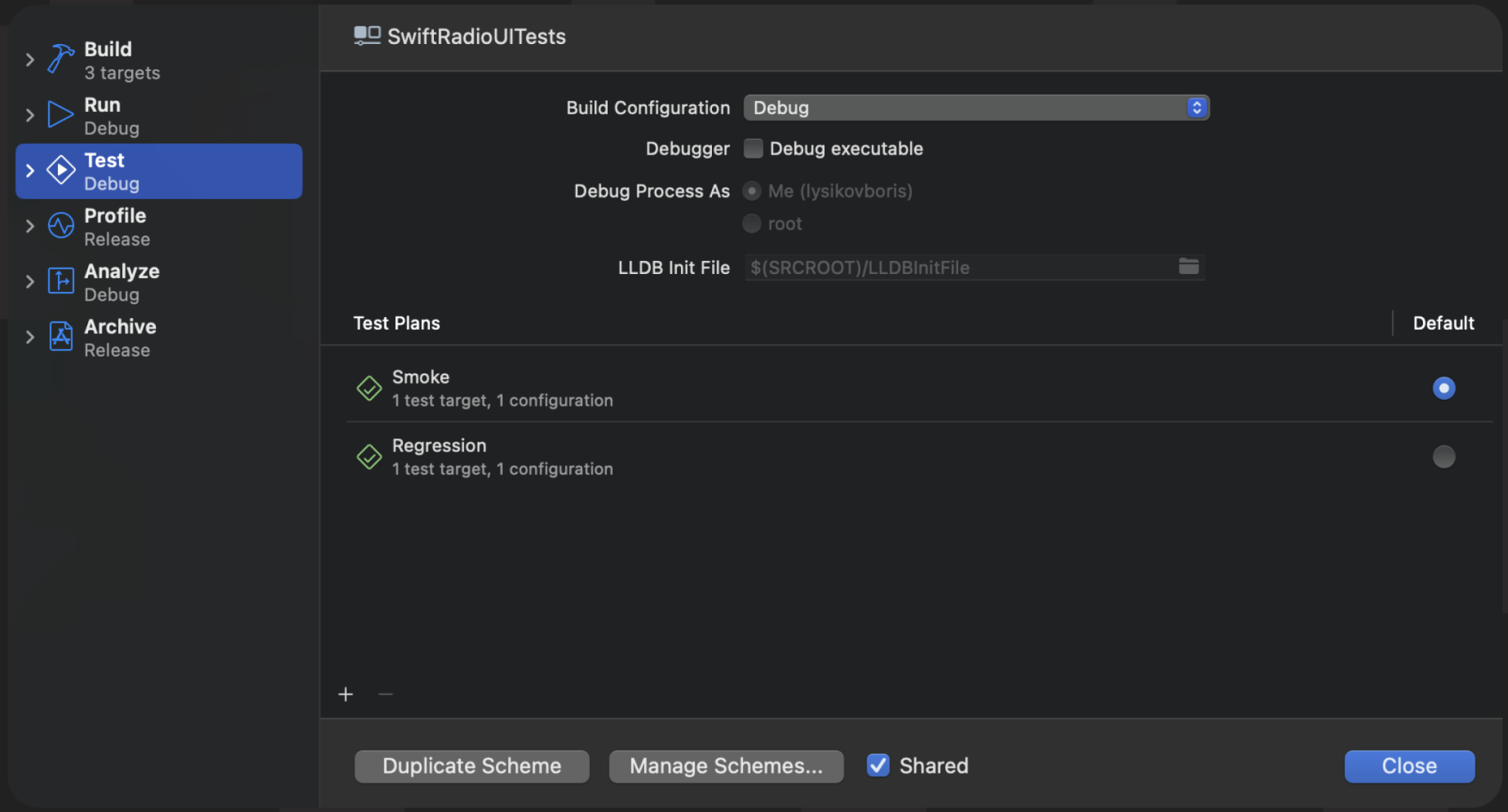
```
xcodebuild test \
  -project SwiftRadio.xcodeproj \
  -scheme SwiftRadioUITests \
  -sdk iphonesimulator \
  -destination 'platform=iOS Simulator,name=iPhone 14 Pro,OS=16.2' \
  -test-timeouts-enabled YES \
  -maximum-test-execution-time-allowance 60 \
  -only-testing SwiftRadioUITests/RadioInfoTest/testClosePage
```

В тест-плане

Как можно сократить

1. **Анализ текущих автотестов**
2. **Убираем компиляцию проекта**
3. **Оптимизация флоу ui-тестов**
    a. Поднимаем экраны
    b. Используем моки
    c. Делаем ограничение на время прохождения автотеста
    d. Отключение debugExecutable

Отключить debugExecutable



Как можно сократить

1. **Анализ текущих автотестов**
2. **Убираем компиляцию проекта**
3. **Оптимизация флоу ui-тестов**
    a. Поднимаем экраны
    b. Используем моки
    c. Делаем ограничение на время прохождения автотеста
    d. Отключение debugExecutable
    e. Выключение анимации

Выключить анимацию

UIKit

```
func application(_ application: UIApplication,
                 didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
    #if DEBUG
    UIView.setAnimationsEnabled(false)
    #endif
    return true
}
```

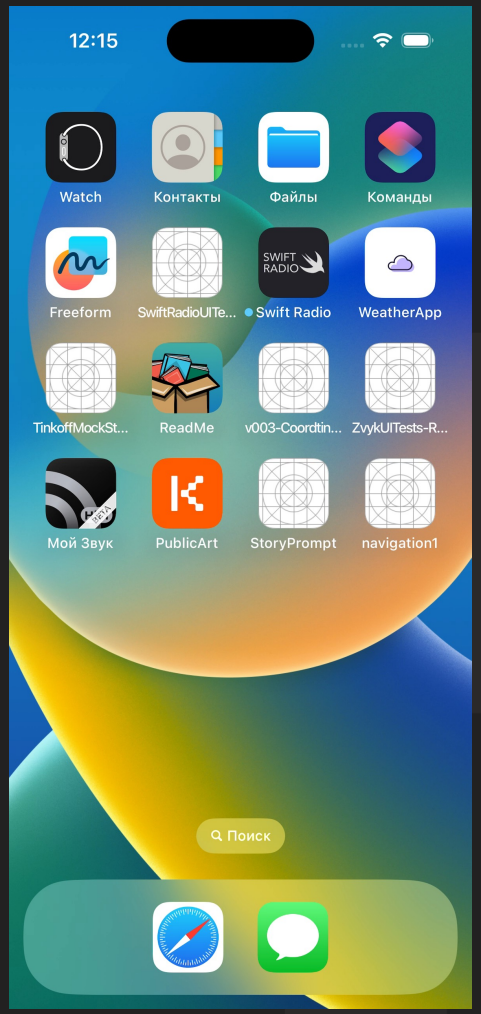
Выключить анимацию

SwiftUI

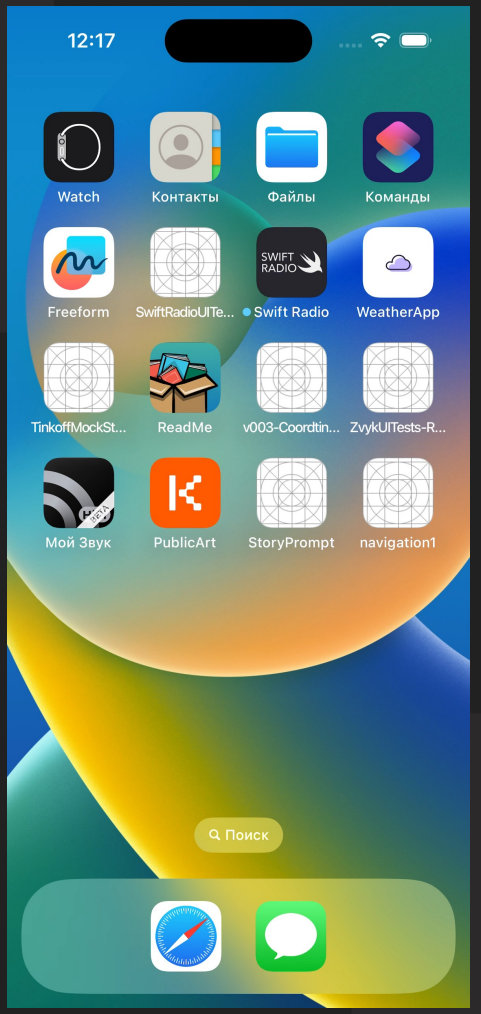
```
#if DEBUG
.transaction { transaction in
    transaction.animation = nil
}
#endif
```

Выключить анимацию

Включенная анимация



Выключенная анимация



Как можно сократить

1. **Анализ текущих автотестов**
2. **Убираем компиляцию проекта**
3. **Оптимизация флоу ui-тестов**
    a. Поднимаем экраны
    b. Используем моки
    c. Делаем ограничение на время прохождения автотеста
    d. Отключение debugExecutable
    e. Выключение анимации
    f. Отказ от sleep

Отказ от sleep

~~Thread.sleep(forTimeInterval: 5)~~

element.waitForExistence(timeout: 5)

Как можно сократить

1. **Анализ текущих автотестов**
2. **Убираем компиляцию проекта**
3. **Оптимизация флоу ui-тестов**
    a. Поднимаем экраны
    b. Используем моки
    c. Делаем ограничение на время прохождения автотеста
    d. Отключение debugExecutable
    e. Выключение анимации
    f. Отказ от sleep
4. Параллелизация тестов

На чем запускать тесты?

1. Симуляторы

На чем запускать тесты?

1. Симуляторы
2. Реальные устройства

Плюсы и минусы симулятора

Плюсы

- Доступность
- Широкий выбор устройств
- Иногда быстрее чем реальные устройства
- Масштабируемость

Минусы

- Ограниченные аппаратные возможности
- Недостоверное поведение

Плюсы и минусы реального устройства

Плюсы

- Более точные результаты
- Доступность всех аппаратных особенностей

Минусы

- Затраты на закуп
- Иногда дольше чем симуляторы
- Высокий риск вздутия аккумулятора

Где запускать?

1. Своё железо (Mac mini)

Где запускать?

1. Своё железо (Mac mini)
2. Облачные решения (xcode cloud, marathon cloud)

Где запускать?

1. Своё железо (Mac mini)
2. Облачные решения (xcode cloud, marathon cloud)
3. Фермы устройств (aws device farm, firebase)

Какие есть раннеры для запуска тестов параллельно

Раннер / фичи	умение работать с тест планами	уникальные uuid при клонировании симулятора	параллелизация на несколько машин	объединение нескольких xcresults	кросс-платформенность
xcodebuild	да	нет	нет	нет	нет
fastlane	да	нет	нет	нет	да
marathon	нет	да	да	нет	да
emcee	да	да	да	да	нет

Какие есть раннеры для запуска тестов параллельно

Раннер / фичи	умение работать с тест планами	уникальные uuid при клонировании симулятора	параллелизация на несколько машин	объединение нескольких xcresults	кросс-платформенность
xcodebuild	да	нет	нет	нет	нет
fastlane	да	нет	нет	нет	да
marathon	нет	да	да	нет	да
emcee	да	да	да	да	нет

Какие есть раннеры для запуска тестов параллельно

Раннер / фичи	умение работать с тест планами	уникальные uuid при клонировании симулятора	параллелизация на несколько машин	объединение нескольких xcresults	кросс-платформенность
xcodebuild	да	нет	нет	нет	нет
fastlane	да	нет	нет	нет	да
marathon	нет	да	да	нет	да
emcee	да	да	да	да	нет

Какие есть раннеры для запуска тестов параллельно

Раннер / фичи	умение работать с тест планами	уникальные uuid при клонировании симулятора	параллелизация на несколько машин	объединение нескольких xcresults	кросс-платформенность
xcodebuild	да	нет	нет	нет	нет
fastlane	да	нет	нет	нет	да
marathon	нет	да	да	нет	да
emcee	да	да	да	да	нет

Какие есть раннеры для запуска тестов параллельно

Раннер / фичи	умение работать с тест планами	уникальные uuid при клонировании симулятора	параллелизация на несколько машин	объединение нескольких xcresults	кросс-платформенность
xcodebuild	да	нет	нет	нет	нет
fastlane	да	нет	нет	нет	да
marathon	нет	да	да	нет	да
emcee	да	да	да	да	нет

Какие есть раннеры для запуска тестов параллельно

Раннер / фичи	умение работать с тест планами	уникальные uuid при клонировании симулятора	параллелизация на несколько машин	объединение нескольких xcresults	кросс-платформенность
xcodebuild	да	нет	нет	нет	нет
fastlane	да	нет	нет	нет	да
marathon	нет	да	да	нет	да
emcee	да	да	да	да	нет

Результаты

~~4 часа 12 минут~~

37 минут

КОМАНДА ЗВУКА

Спасибо за внимание!

тг-канал

Материалы

Борис Лысиков

Руководитель отдела автоматизированного тестирования