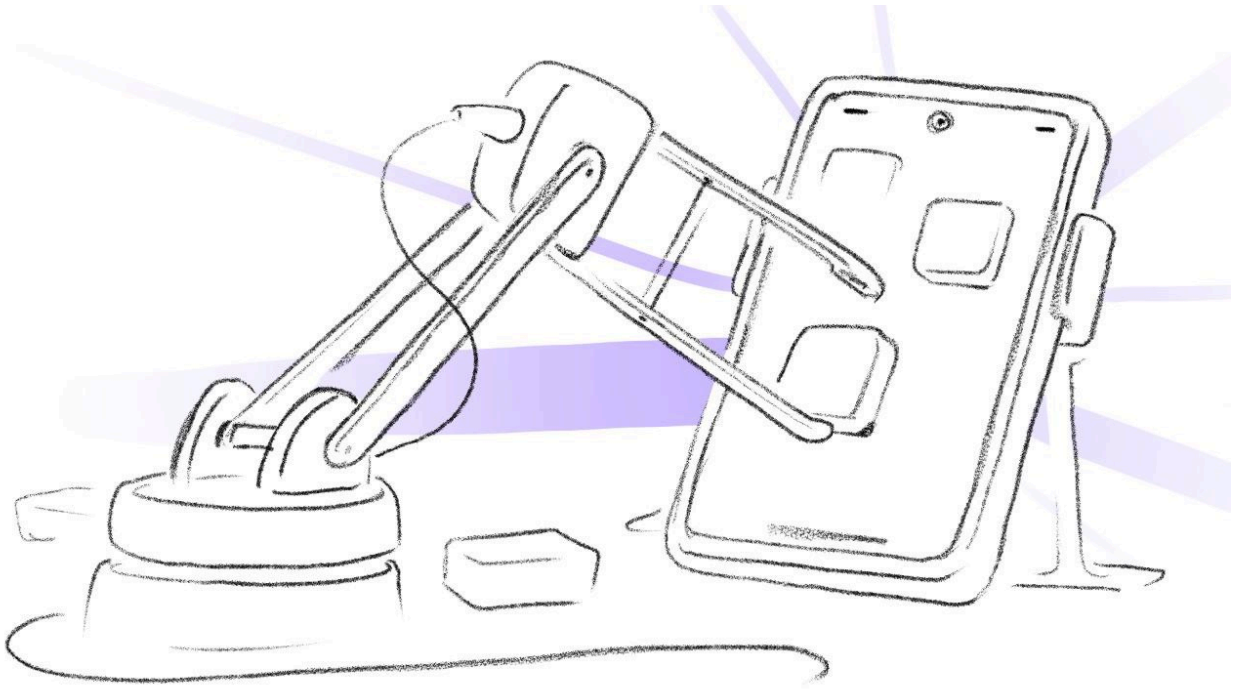


Внедрение Allure (отчетность) в UI тесты (swift, XCTest)

Меня зовут Сергей, я занимаюсь нативной автоматизацией тестирования мобильных приложений (iOS) в компании «РТ-Лабс».

В данной статье я расскажу, как внедрялась отчетность на наших проектах, какие средства использовались и о порядке действий для получения результатов прогона автотестов в **Allure**.

Краткое содержание статьи:

1. Введение

Мы используем следующий стек инструментов для создания UI-тестов мобильного приложения для iOS и прогона тестовых наборов:
- Swift;
- XCTest;
- XCode;
- Jenkins.

Наш отдел выполняет тестирование мобильных приложений на продуктивной среде (проде) ежедневно. После внедрения UI-тестов мануальным тестировщикам требовалась

исчерпывающая информация о том, как прошел смоук-набор автотестов, в связи с чем возникла потребность подобрать оптимальный вариант предоставления отчета для них.

Стандартный отчет ***.xctestresult**, который генерирует **XCode**, не предоставлял им полной и подробной информации для обработки результатов при разборе прогона.

Необходимо было, чтобы отчет содержал название тест-кейса, ссылку на кейс, шаги кейса, скриншоты, а также был простым и наглядным.

2. Анализ подобранных вариантов ведения отчетности

При выборе решения в качестве вариантов были рассмотрены плагины **fastlane-plugin-test_center**, **XCTestHTMLReport** и **xcresults** (в связке с **Allure**).

Наиболее подходящим решением стал плагин **xcresults** (в связке с **Allure**), так как информация, предоставленная им, имеет самый расширенный функционал, такой, как:
- процентное соотношение пройденных / проваленных / скипнутых и др. тестов;
- таймлайн;
- графики с количественной статистикой по статусам, длительности, ретраям и др.;
- подробное отображение информации по каждому тесту (название теста, ссылка на кейс, шаги, скриншот, длительность теста и каждого шага, ретраи и др.)

Помимо перечисленного, **Allure** является основным решением при автоматизации других направлений (web, API), поэтому будет удобен и привычен при использовании и на мобильном приложении.

Приведу краткое описание и итоговый результат для промежуточных вариантов.

2.1. Вариант fastlane-plugin-test_center

Для внедрения **fastlane-plugin-test_center** необходимо выполнить установку **fastline** и плагина **fastlane-plugin-test_center**, сконфигурировать файл **Fastfile** (более подробная инструкция в документации https://docs.fastlane.tools/), а затем запустить прогон автотестов.

Для этого в **cmd** нужно выполнить следующую команду в директории проекта:

```
fastlane test
```

где *test* – имя *lane* из файла **Fastfile**

Итоговый отчет выглядит следующим образом:

*

* Изображение взято с сайта https://docs.fastlane.tools/actions/scan/

Данный отчет содержит минимальную информацию, такую, как кол-во пройденных / упавших тестов, результат выполнения теста, названия тестов (но не самих кейсов), шаги, длительность тестов.

2.2. Вариант XCTestHTMLReport

1. Для подключения **XCTestHTMLReport** необходимо установить плагин.

Для этого в **cmd** нужно выполнить следующую команду:

```
brew install XCTestHtmlReport/xchtmlreport/xchtmlreport
```

2. Перейти в директорию проекта

Для этого в **cmd** нужно выполнить следующую команду:

```
cd pathToProject
```

3. Запустить прогон автотестов.

Например, выполнив в **cmd** команду:

```
xcodebuild test -workspace workspacename.xcworkspace -scheme schemename -destination 'platform=iOS Simulator,name=iPhone 12,OS=15.2' -testPlan NameTestPlan -resultBundlePath TestResults
```

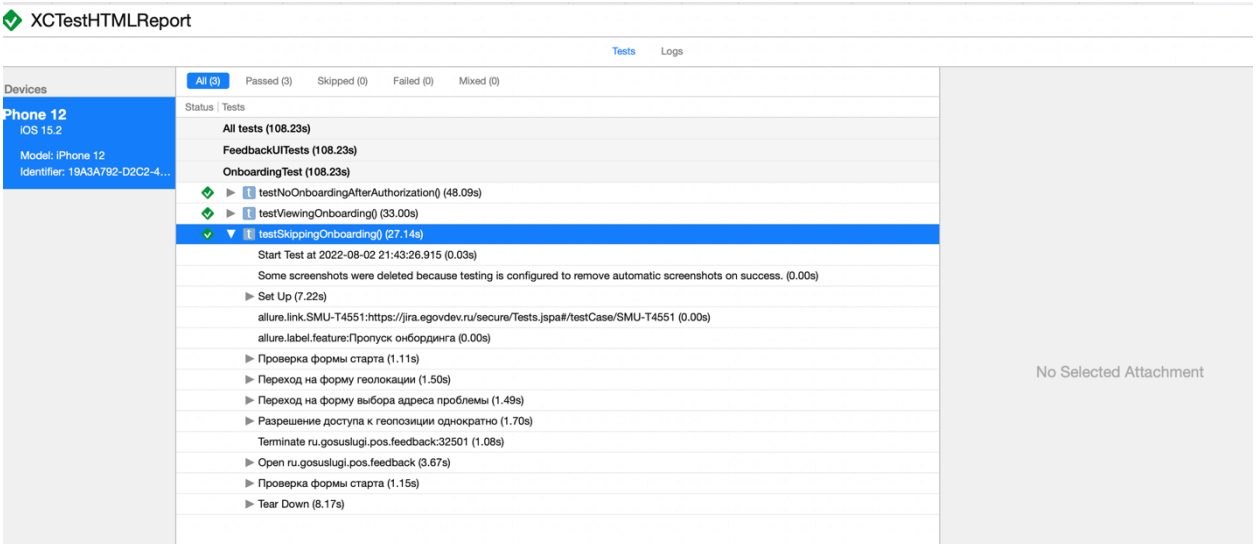
4. Сформировать отчет.

Для этого в **cmd** нужно выполнить следующую команду:

```
xchtmlreport -r TestResults
```

5. Открыть отчет – файл index.html.

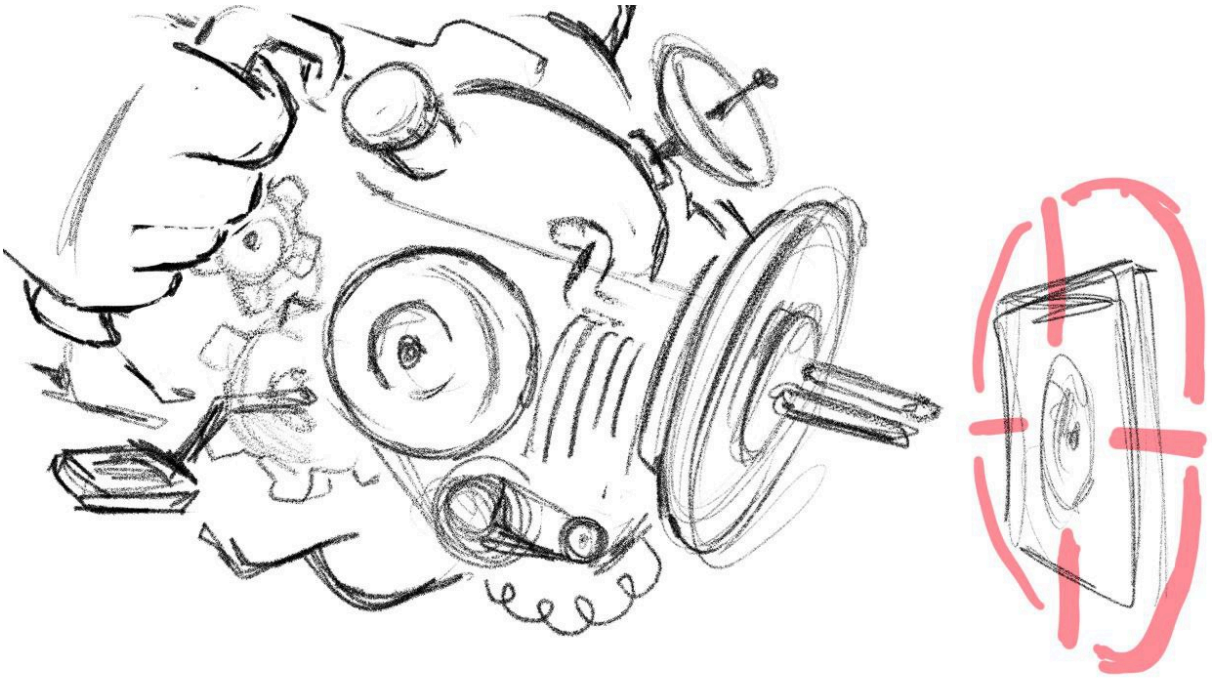
Итоговый отчет выглядит следующим образом:



Полученный вариант более информативный, чем первый. Отчет содержит данные о кол-ве пройденных / упавших тестов, результат выполнения теста, названия тестов (но не самих кейсов), шаги, длительность тестов, сведения об устройстве запуска, скриншоты.

2.3. Вариант xcresults (в связке с Allure)

Более подробно остановимся на порядке действий для подключения **Allure**-отчета, так как он был выбран в качестве оптимального ведения отчетности из-за возможности предоставлять максимально подробную информацию.



3. Подключение Allure

3.1. Внедрение аннотаций

1. Для начала необходимо подготовить аннотации в коде проекта.

Для этого нужно создать новый файл в модуле с автотестами:

```
8
9  import XCTest
10
11 extension XCTest {
12     func feature(_ values: String...) {
13         label(name: "feature", values: values)
14     }
15
16     private func label(name: String, values: [String]) {
17         for value in values {
18             XCTContext.runActivity(named: "allure.label." + name + ":" + value, block: { _ in })
19         }
20     }
21
22     func link(_ value: String) {
23         links(name: value, values: ["https://jira.*.ru/testCase/" + value])
24     }
25
26     private func links(name: String, values: [String]) {
27         for value in values {
28             XCTContext.runActivity(named: "allure.link." + name + ":" + value, block: { _ in })
29         }
30     }
31 }
32
```

Метод *feature* используется для названия тест-кейса.
Метод *link* используется для указания ссылки на тест-кейс.
Методы *label* и *links* – вспомогательные.

Аналогично можно в данный файл можно добавить методы *id*, *epic*, *story* и др. для указания дополнительной информации.

Например:

```
16      func story(_ stories: String...) {
17          label(name: "story", values: stories)
18      }
```

2. Следующим шагом является добавление аннотаций в автотесты:

```
9  import Foundation
10
   class AuthTest: Base {
       func test_example() {
13         link("S-T1")
14         feature("Авторизация по номеру телефона")
15         print("Run test...")
16     }
17 }
18
```

3. Чтобы в отчете отображалось названия шагов, есть два способа:

1 способ

В файл с аннотациями добавить метод *step*:

```
15
16      func step(_ name: String, step: () -> Void) {
17          XCTContext.runActivity(named: name) { _ in
18              step()
19          }
20      }
```

В автотесте использовать аннотацию *step*:

```
8
9  import Foundation
10 import XCTest
11
   class AuthTest: Base {
       func test_example() {
14         link("T-123")
15         feature("Авторизация в МП по номеру телефона")
16         step("Проверка отображения кнопки 'TestButton'") {
17             XCTAssertTrue(app.buttons["TestButton"].exists)
18         }
19         step("Тап по кнопке 'TestButton'") {
20             app.buttons["TestButton"].tap()
21         }
22     }
23 }
```

* Примечание: в данном примере для наглядности проверки осуществляются непосредственно в методе теста, на самом деле корректно выносить их в отдельный файл и вызывать методы проверки в методе теста.

2 способ

В файлах с методами проверок использовать XCTContext.runActivity:

```
@discardableResult
func checkTestButton() -> Self {
    XCTContext.runActivity(named: "Проверка отображения кнопки 'TestButton'") { _ in
        XCTAssertTrue(app.buttons["TestButton"].exists)
    }
    XCTContext.runActivity(named: "Тап по кнопке 'TestButton'") { _ in
       app.buttons["TestButton"].tap()
    }
    return self
}
```

4. Чтобы в отчет попадали скриншоты, необходимо в методе tearDown вызывать метод, сохраняющий скриншоты, например, takeScreenShot:

```swift
override func tearDown() {
    super.tearDown()
    takeScreenShot()
}

func takeScreenShot(name screenshotName: String? = nil) {
    let screenshot = XCUIScreen.main.screenshot()
    let attach = XCTAttachment(screenshot: screenshot, quality: .original)
    attach.name = screenshotName ?? name + "_" + UUID().uuidString
    attach.lifetime = .keepAlways
    add(attach)
}
```

3.2. Установка необходимых компонентов

Для локального запуска автотестов понадобится:

1. Установить **Allure**

Для этого в **cmd** нужно выполнить следующую команду:

```
brew install allure
```

2. Установить **xcresults**

Для этого в **cmd** нужно выполнить следующую команду:

```
wget https://github.com/eroshenkoam/xcresults/releases/latest/download/xcresults
```

Если не установлен **wget**, установить его, выполнив команду:

```
brew install wget
```

3. Сделать исполняемым установленный файл

Для этого в **cmd** нужно выполнить следующую команду:

```
chmod +x xcresults
```

Для запуска автотестов на Jenkins:

Добавить shell-команды в разделе “Сборка”:

```
wget https://github.com/eroshenkoam/xcresults/releases/latest/download/xcresults
chmod +x xcresults
```

3.3. Формирование отчета

Локально

1. Перейти в директорию проекта

Для этого в **cmd** нужно выполнить следующую команду:

```
cd pathToProject
```

2. Запустить прогон автотестов

Например, выполнив в **cmd** команду:

```
xcodebuild test -workspace workspacename.xcworkspace -scheme schemename -destination 'platform=iOS Simulator,name=iPhone 12,OS=15.2' -testPlan NameTestPlan -resultBundlePath TestResults
```

По итогу должно быть получено два файла **TestResult** и **TestResult.xctestresult**.

3. Создать директорию для формирования отчета

Для этого в **cmd** нужно выполнить следующую команду:

```
mkdir allure-results
```

4. Запустить экспорт xctestresult-отчета в allure-отчет

Для этого в **cmd** нужно выполнить следующую команду:

```
xcresults export TestResults.xcresult allure-results
```

Если будет получено ***command not found: xcresults***, заменить команду на следующую:

```
./xcresults export TestResults.xcresult allure-results
```

5. Запустить генерацию Allure-отчета:

Для этого в **cmd** нужно выполнить следующую команду:

```
allure serve allure-results
```

Для Jenkins:

После команды запуска автотестов
Например:

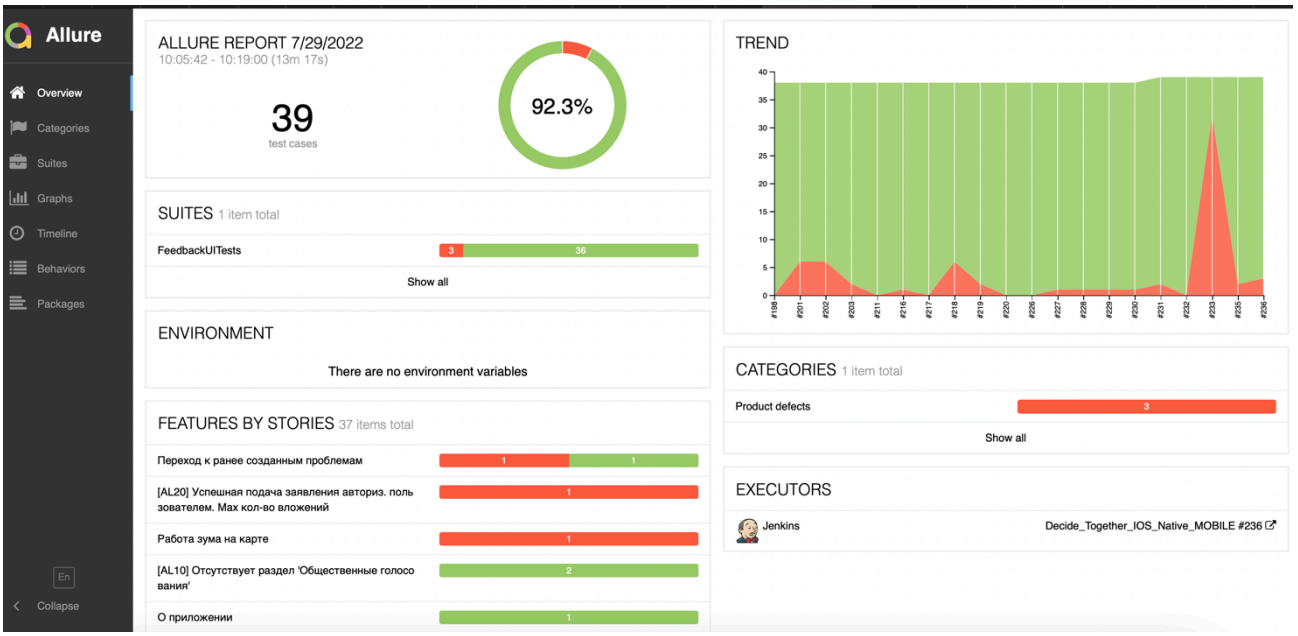
```
xcodebuild test -workspace workspacename.xcworkspace -scheme schemename -destination 'platform=iOS Simulator,name=iPhone 12,OS=15.2' -testPlan NameTestPlan -resultBundlePath TestResults
```

Добавить shell-команды в разделе “Сборка”:

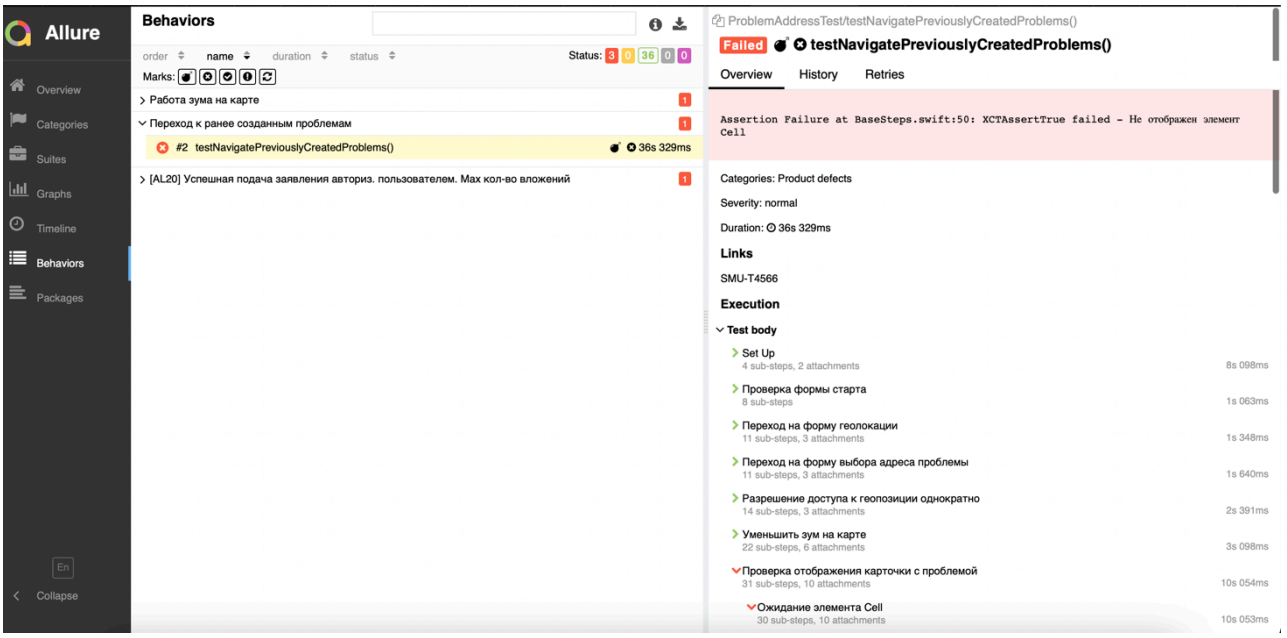
```
mkdir allure-results
xcresults export TestResults.xcresult allure-results
```

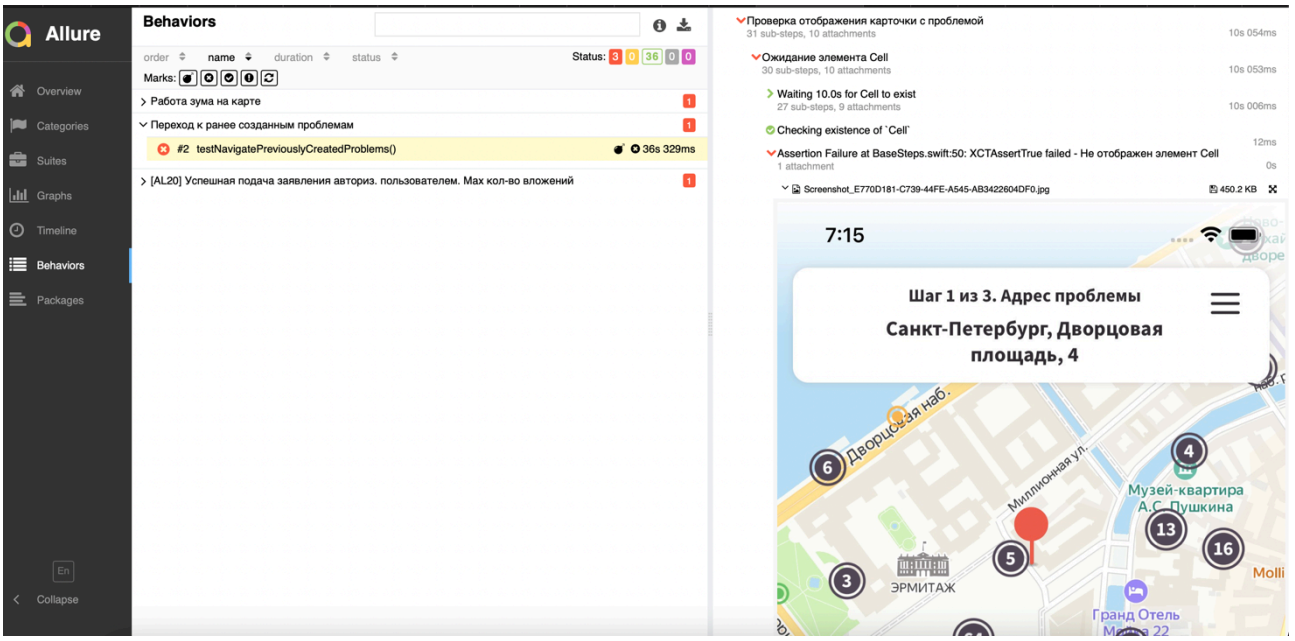
3.4. Анализ отчета

После выполнения всех действий будет получен отчет:



В него попадают название теста, ссылка на тест-кейс, шаги и скриншот падения:





Как видно из скриншотов, информация **Allure**-отчета более содержательная и ее удобно анализировать.

Внедрение отчета предоставило ручным тестировщикам удобный переход по ссылке непосредственно к тест-кейсу, отображение название кейса (а не теста), простую навигацию по набору тестов, подробную информацию о прохождении (шаги, ошибки при падении, скриншоты и др.).
Для автоматизаторов выбранное решение дало дополнительные сведения для анализа в виде таймлайнов и других сборных данных по прогону, а руководители могут наглядно увидеть краткий результат в виде диаграммы с процентом пройденных / проваленных автотестов и времени прогона.

4. Заключение

В дальнейшем мы планируем переход к **TestOps**'у, и выбранное решение для отчетности в виде использования плагина **xcresults** (в связке с **Allure**) также будет актуальным.

Пишите в комментариях, какой вариант используете вы на своих проектах, или с какими интересными решениями вам довелось сталкиваться.

Спасибо за внимание!

Криворотько Сергей, ведущий инженер по тестированию, «РТ Лабс».