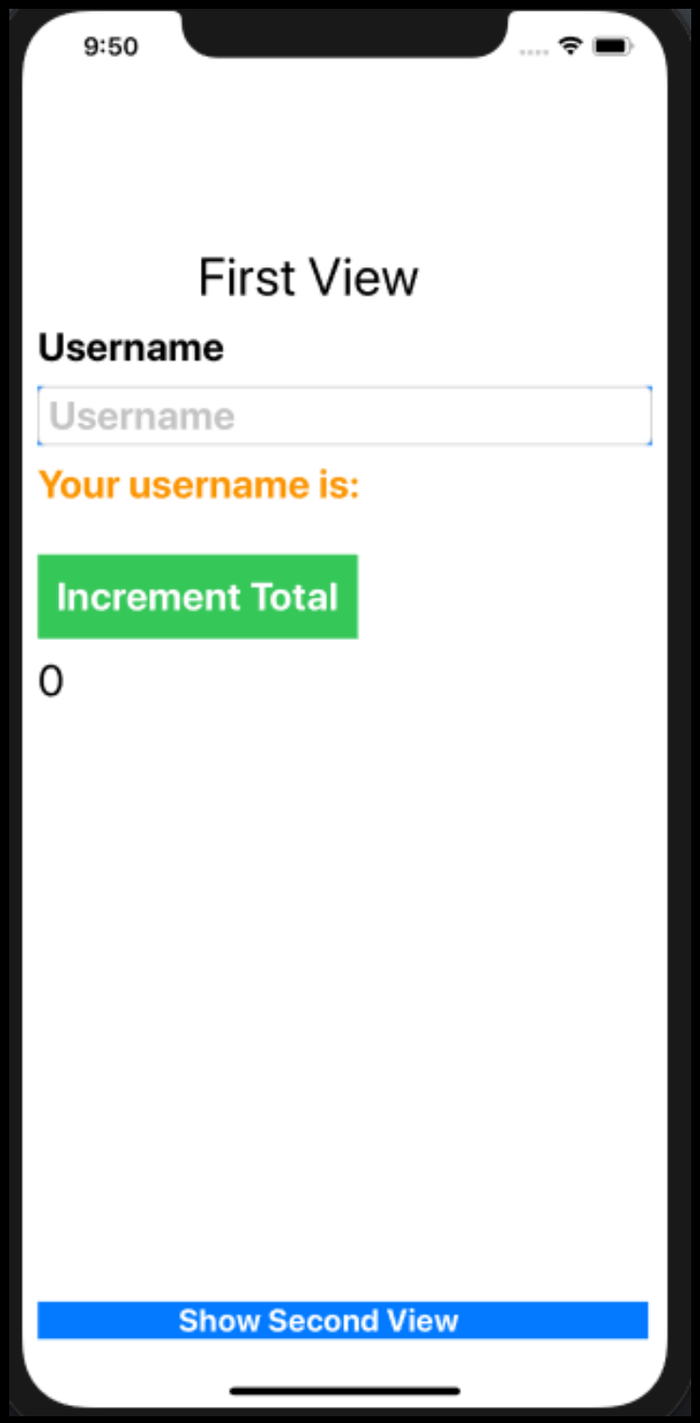
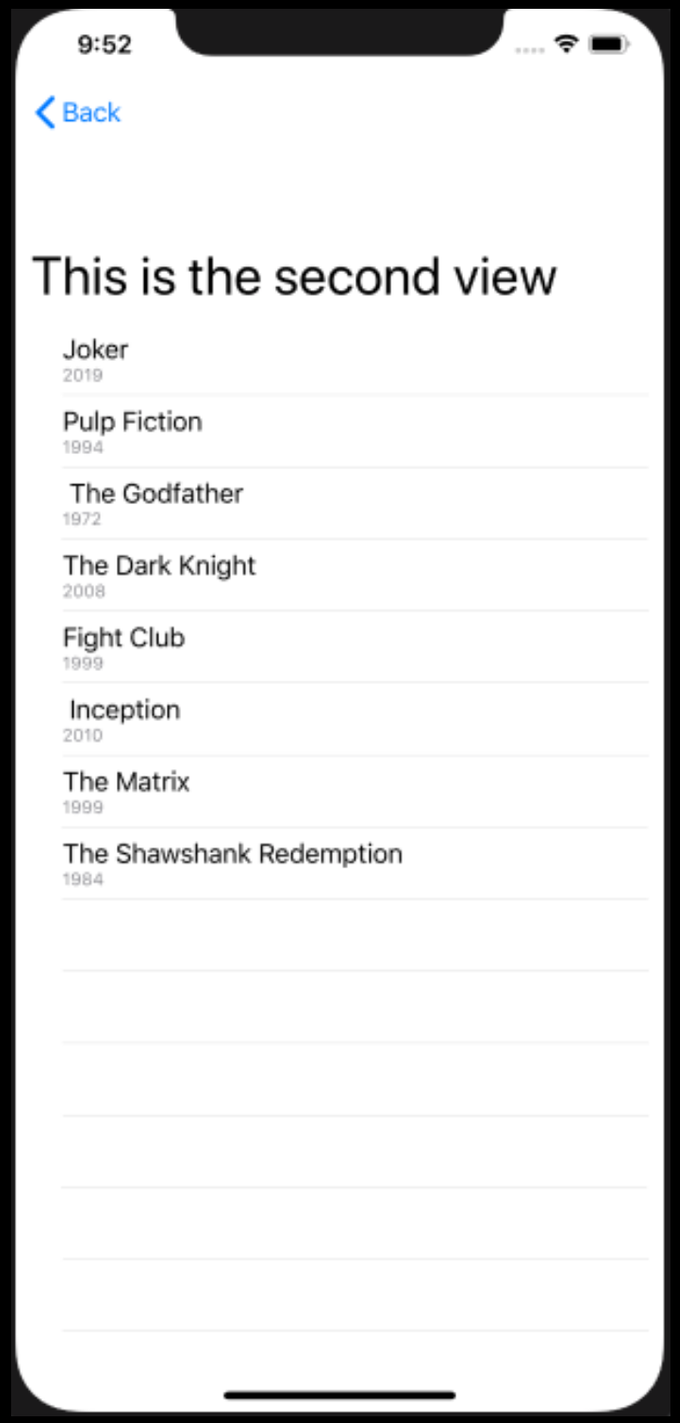
I put together some slides to show you how I architect the XCUITest framework for an iOS project.

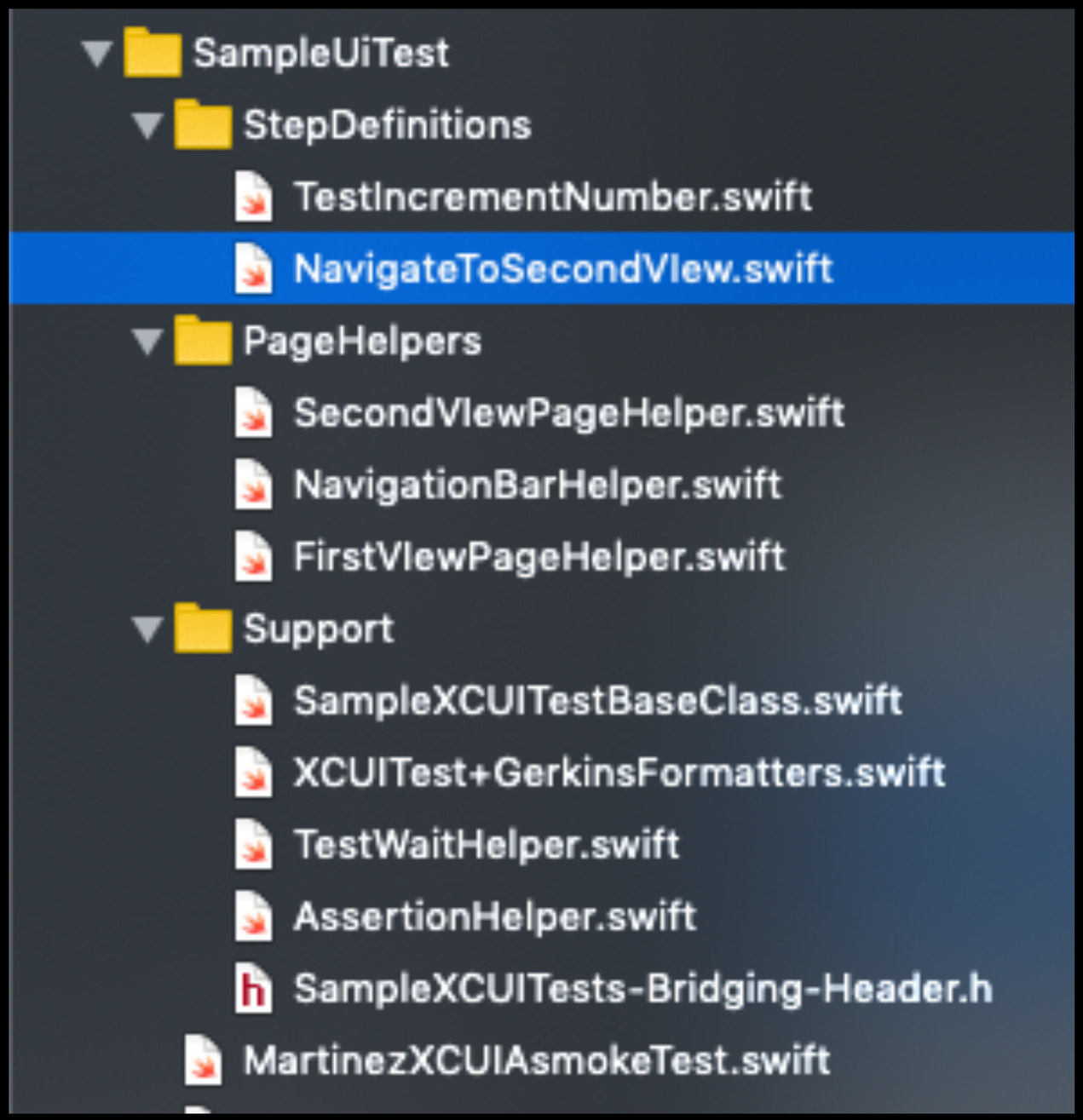
Eric Martinez

Using swiftUi I create a simple app with two views.





XUITest file structure

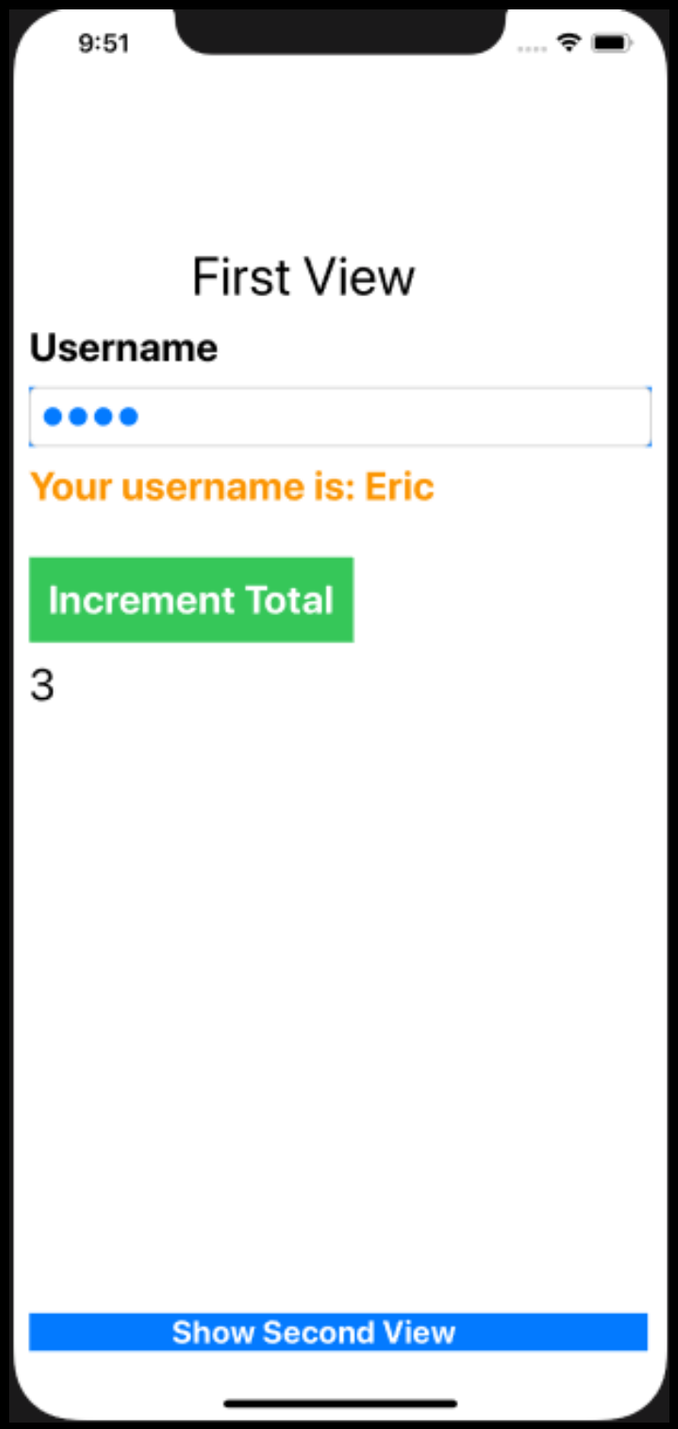


Created a class for the different test scenarios

I wrote a smoke test using Gherkin's syntax.

```
class MartinezXCUIAsmokeTest: MartinezBaseXCUITestHelper {

    func testIncrementTotal() {

        given("I launch the App I validate all first view UI element appear") {
            incrementScreenElementValidation()
        }

        when("I enter the default username for testing") {
            typeDefaultUsernanme()
        }

        then("I increment the number to three") {
            incrementNumberTest()
        }

        and("I validate that the number has been increased") {
            defaultDataPersist()
        }
    }

    func testNavigateToSeconview() {

        given("I'm on the First View") {
            firstViewValidation()
        }

        when("I navigate to the second view") {
            navigateToSecondView()
        }

        then("I validate that the list of movies appear") {
            validateMoviesInTableView()
        }

        and("I navigate back to the first view") {
            navigateBackToFirstView()
        }
    }
}
```

First scenario



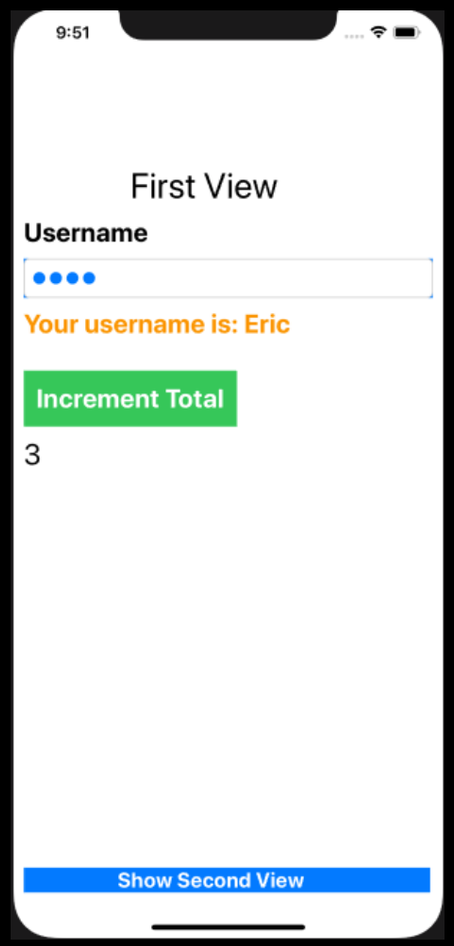
First view step definitions

In this file you will nest the necessary functions, and methods from different pages to complete the desired user journey.

```swift
//  Created by Eric Martinez on 1/17/20.
//  Copyright © 2020 emobile. All rights reserved.
//

import Foundation
import XCTest

extension XCTestCase {
    func typeDefaultUsernanme() {
        application.tapfirstViewTextField(.userName)
        application.typeText(FirstVIewPageHelper.defaultNameEntry.rawValue)
        application.buttons["Return"].tap() // Soft keyboard event needs its own helper
        assertTrueStaticTexts(text: FirstVIewPageHelper.defaultNameEntryResult.rawValue)
    }

    func incrementScreenElementValidation() {
        assertTrueStaticTexts(text: FirstVIewPageHelper.defaultNumber.rawValue)
        assertTrueButtons(name: FirstVIewPageHelper.incrementTotal.rawValue)
    }

    func incrementNumberTest() {
        application.tapfirstViewButton(.incrementTotal)
        application.tapfirstViewButton(.incrementTotal)
        application.tapfirstViewButton(.incrementTotal)
        waiting(for: application.staticTexts["3"])
    }

    func defaultDataPersist() {
        application.tapfirstViewButton(.showSecondView)
        self.waitForElementToAppear(NavigationBarHelper.navBarBackButton.navBarButton(in: application.self)!)
        application.tapNavBarButton(.navBarBackButton)
        assertTrueStaticTexts(text: FirstVIewPageHelper.defaultNameEntryResult.rawValue)
        waiting(for: application.staticTexts["3"])
    }
}
```

First view test page helper

When there are changes in the view this format makes it easy to update



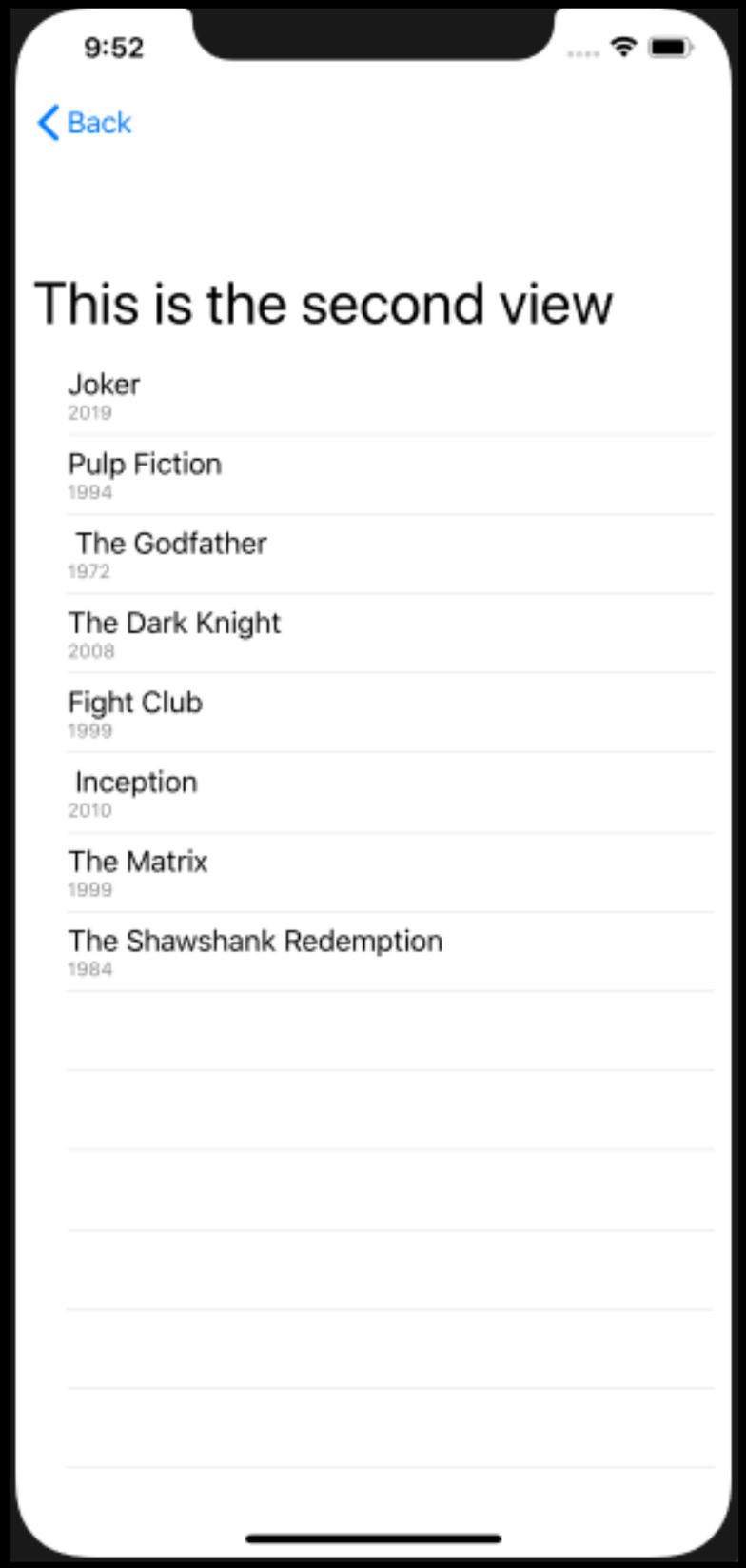

```swift
enum FirstVIewPageHelper: String {
    case viewTitle = "                First View"
    case showSecondView = "                        Show Second View                                "
    case incrementTotal = "Increment Total"
    case defaultNumber = "0"
    case userName = "Username"
    case defaultNameEntry = "Eric"
    case defaultNameEntryResult = "Your username is: Eric"

    func firstViewStaticTexts(in application: XCUIApplication) -> XCUIElement? {
        return application.staticTexts[self.rawValue]
    }

    func firstViewButton(in application: XCUIApplication) -> XCUIElement? {
        return application.buttons[self.rawValue]
    }

    func firstViewTextField(in application: XCUIApplication) -> XCUIElement? {
        return application.secureTextFields[self.rawValue]
    }
}

extension XCUIApplication {
    func tapfirstViewStaticTexts(_ firstViewStaticTexts: FirstVIewPageHelper) {
        guard let element = firstViewStaticTexts.firstViewStaticTexts(in: self) else {
            XCTFail("failed to tap static text in Sign In screen")
            return
        }
        element.tap()
    }

    func tapfirstViewButton(_ firstViewButton: FirstVIewPageHelper) {
        guard let element = firstViewButton.firstViewButton(in: self) else {
            XCTFail("failed to tap button in Sign In screen")
            return
        }
        element.tap()
    }

    func tapfirstViewTextField(_ firstViewTextField: FirstVIewPageHelper) {
        guard let element = firstViewTextField.firstViewTextField(in: self) else {
            XCTFail("failed to tap text field in Sign In screen")
            return
        }
        element.tap()
    }
}
```

Navigation bar Helper

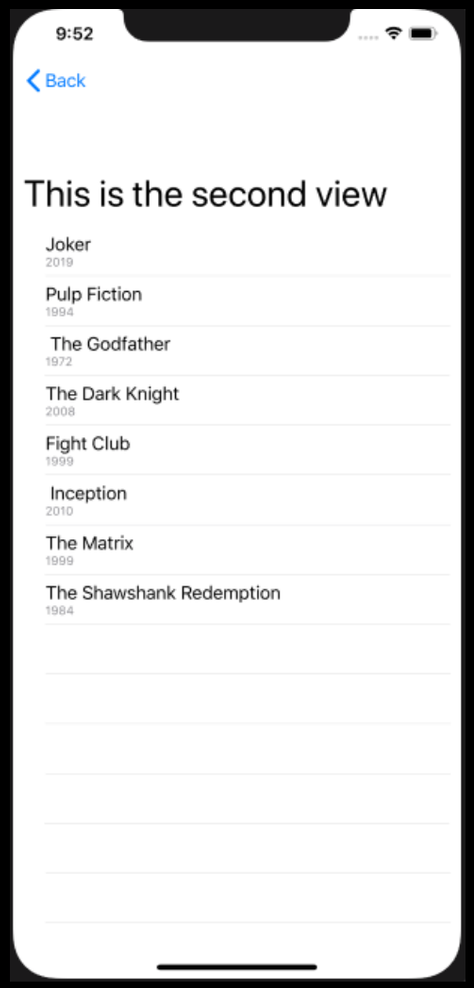
The navBar can appear on all pages with the same buttons so I feel is best to give it its own page helper and use it across all test.

```swift
//
//  NavigationBarHelper.swift
//  SampleUiTest
//
//  Created by Eric Martinez on 1/18/20.
//  Copyright © 2020 emobile. All rights reserved.
//

import Foundation
import XCTest

enum NavigationBarHelper: String {

    case navBarBackButton = "Back"

    func navBarButton(in application: XCUIApplication) -> XCUIElement? {
        return application.buttons[self.rawValue]
    }
}

extension XCUIApplication {

    func tapNavBarButton(_ navBarButton: NavigationBarHelper) {
        guard let element = navBarButton.navBarButton(in: self) else {
            XCTFail("failed to tap Navigation bar button")
            return
        }
        element.tap()
    }
}

```

Navigate to second view test



Second view page helper



You can write loops to iterate over Ui elements on the view or other functions related the page

```swift
enum SecondViewPageHelper: String {
    case secondViewTitle = "This is the second view"
    case propertyJoker = "Joker"
    case propertyJokerRelease = "2019"

    func secondViewStaticText(in application: XCUIApplication) -> XCUIElement? {
        return application.staticTexts[self.rawValue]
    }

    func secondViewTableStaticText(in application: XCUIApplication) -> XCUIElement? {
        return application.tables.staticTexts[self.rawValue]
    }
}

extension XCUIApplication {

    func tapSecondViewStaticText(_ secondViewStaticText: SecondViewPageHelper) {
        guard let element = secondViewStaticText.secondViewStaticText(in: self) else {
            XCTFail("failed to tap Static text")
            return
        }
        element.tap()
    }

    func findSecondViewTableStaticText(_ secondViewTableStaticText: SecondViewPageHelper) {
        guard let element = secondViewTableStaticText.secondViewTableStaticText(in: self) else {
            XCTFail("failed to find Table Static text")
            return
        }
        element.tap()
    }
}

func assertTrueTablesStaticText(_ movieInTable: String) {
    XCTAssertTrue(application.tables.staticTexts[movieInTable].exists)
}

func movieListInTable() {
    let moviesList = ["Joker", "Pulp Fiction", " The Godfather ", "The Dark Knight ",
                      "Fight Club", " Inception", "The Matrix ", "The Shawshank Redemption "]
    print(moviesList)
    for movie in moviesList {
        assertTrueTablesStaticText(movie)
    }
}
```

Second view step definitions

```
//
//  Created by Eric Martinez on 1/18/20.
//  Copyright © 2020 emobile. All rights reserved.
//

import Foundation
import XCTest

extension XCTestCase {

    func firstViewValidation() {
        self.waitForElementToAppear(FirstVIewPageHelper.showSecondView.firstViewButton(in: application.self)!)
    }

    func navigateToSecondView() {
        application.tapfirstViewButton(.showSecondView)
        self.waitForElementToAppear(NavigationBarHelper.navBarBackButton.navBarButton(in: application.self)!)
        assertTrueStaticTexts(text: SecondViewPageHelper.secondViewTitle.rawValue)
    }

    func validateMoviesInTableView() {
        movieListInTable()
        application.findSecondViewTableStaticText(.propertyJoker)
        application.findSecondViewTableStaticText(.propertyJokerRelease)
    }

    func navigateBackToFirstView() {
        application.tapNavBarButton(.navBarBackButton)
        self.waitForElementToAppear(FirstVIewPageHelper.showSecondView.firstViewButton(in: application.self)!)
    }
}

```

Makefile commands

make test-ui runs all the test in the Ui test framework

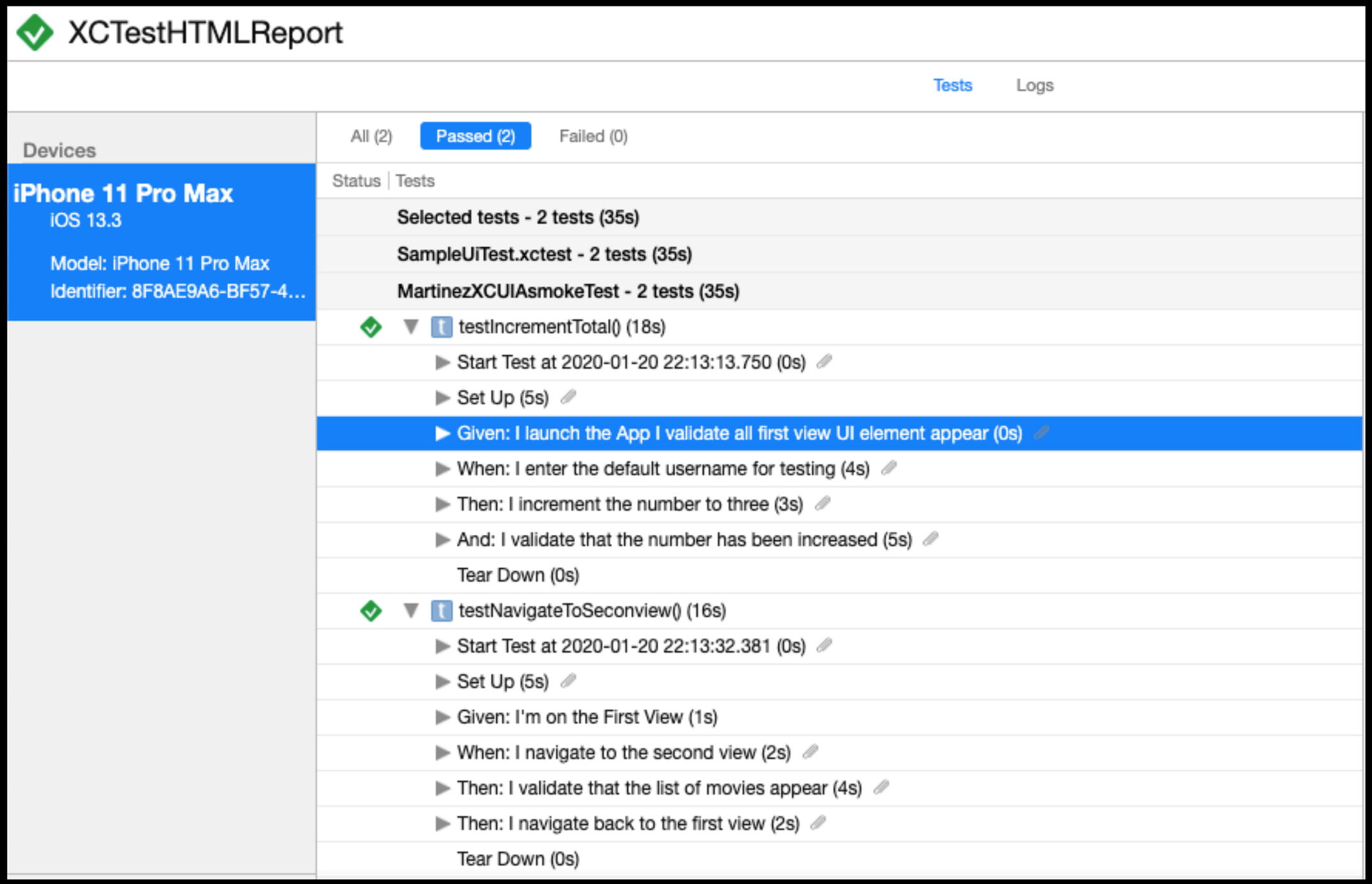
make ui-test-report creates a test report and opens it on a browser

```
94    test-ui:
95        rm -rf TestResults
96        $(XCODEBUILDCMD) test -scheme "SampleUiTest" \
97         -sdk iphonesimulator \
98         -destination '${SIMULATOR_UITEST}' \
99         -resultBundlePath TestResults
100
101   ui-test-report:
102       xcrun simctl shutdown all
103       xchtmlreport -r TestResults
104       echo "Your Test Report test report is launching on our default browser"
105       open index.html
106
```

In CI system we can generate xcpretty reports with colorful graphs for displaying on dashboard

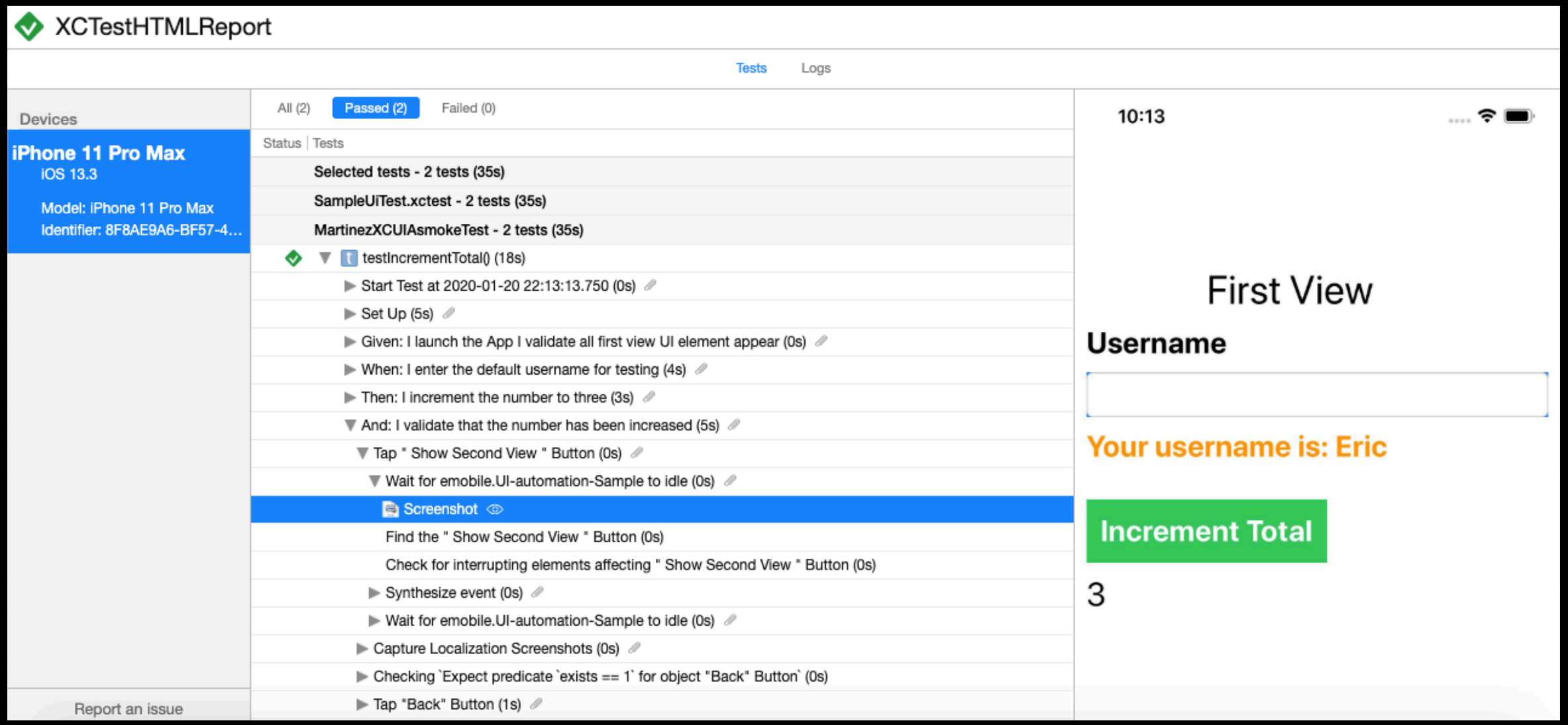
HTML Report

As you can see in this image the gherkins syntax prints out nicely



HTML report

During the run Xcode captures a screenshot for each step



Any questions? ericlmartinez@gmail.com

Thanks!