How to test the Swift in my UI

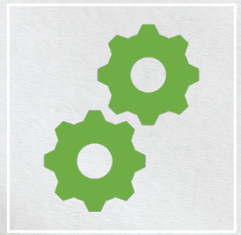
Paulo Santos e Vitor Varela

THE DEVELOPER'S CONFERENCE

The Goal

Agenda

Introduction



Robot Pattern

View Inspector

Snapshot Tests

Agenda

Introduction

Thoughtworks

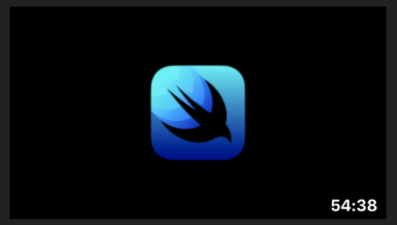
Gmail Imagens Fazer logi

Pesquisa Google Estou com sorte

Disponibilizado pelo Google em: English

rasil

Build great apps in SwiftUI



Introduction to SwiftUI

WWDC 2020

What's new in SwiftUI

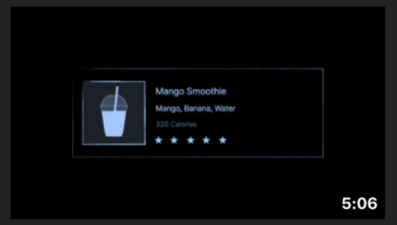
WWDC 2020

Build complications in SwiftUI

WWDC 2020

App essentials in SwiftUI

WWDC 2020



Visually edit SwiftUI views

WWDC 2020

Stacks, Grids, and Outlines in SwiftUI

WWDC 2020

Build a SwiftUI view in Swift Playgrounds

WWDC 2020

Build document-based apps in SwiftUI

WWDC 2020

Data Essentials in SwiftUI

WWDC 2020

Build SwiftUI views for widgets

WWDC 2020

Unit testing in SwiftUI

Asked 10 months ago Active 7 months ago Viewed 1k times

3

I am trying to write unit tests for SwiftUI views but finding zero resources on the web for how to go about that.

I have a view like the following

```
struct Page: View {
@EnvironmentObject var service: Service

var body: some View {
    NavigationView {
        ScrollView(.vertical) {
            VStack {
                Text("Some text"))
                    .font(.body)
                    .navigationBarTitle(Text("Title")))

                Spacer(minLength: 100)
            }
        }
   }
}
}
```

I started writing a test like this

```
func testPage() {
    let page = Page().environmentObject(Service())
    let body = page.body
    XCTAssertNotNil(body, "Did not find body")
}
```

But then how do I get the views inside the body? How do I test their properties? Any help is appreciated.

Update: As a matter of fact even this doesn't work. I am getting the following runtime exception

```
Thread 1: Fatal error: body() should not be called on  ModifiedContent<Page,_Enviro
```

3

Original reply:

Until Apple

a) designs testability into SwiftUI, and

b) exposes this testability to us,

we're screwed, and will have to use UI Testing in place of unit testing… in a complete inversion of the Testing Pyramid.

share improve this answer follow

edited Dec 2 '19 at 0:29

answered Oct 4 '19 at 19:57

Jon Reid

17.9k ● 2 ● 51 ● 82

add a comment

3

There is a framework created specifically for the purpose of runtime inspection and unit testing of SwiftUI views: ViewInspector

So the test for your view would look like this:

```
func testPage() throws {
   let page = Page().environmentObject(Service())
   let string = try page.inspect().navigationView().scrollView()
                        .vStack().text(0).string()
   XCTAssertEqual(string, "Some text")
}
```

share improve this answer follow

edited Jan 12 at 11:54

answered Nov 24 '19 at 18:50

nalexn

8,833 ● 6 ● 37 ● 45

add a comment

3

Update: Let's all try using the ViewInspector library by nalexn!

Original reply:

Until Apple

a) designs testability into SwiftUI, and

b) exposes this testability to us,

we're screwed, and will have to use UI Testing in place of unit testing… in a complete inversion of the Testing Pyramid.

share improve this answer follow

edited Dec 2 '19 at 0:29

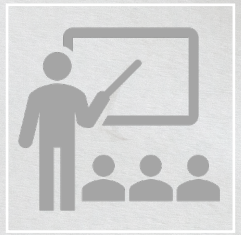
answered Oct 4 '19 at 19:57

Jon Reid

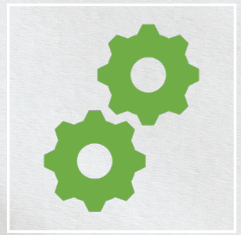
17.9k ● 2 ● 51 ● 82

add a comment

Agenda



Introduction

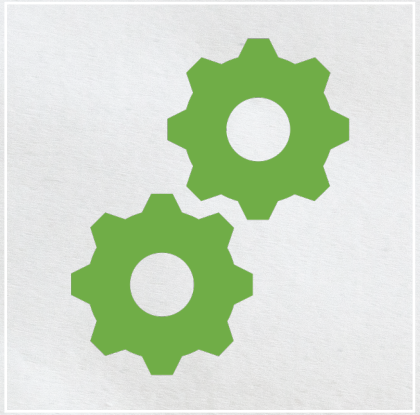


Robot Pattern

View Inspector

Snapshot Tests

Agenda



Robot Pattern

UI Testing

XCUITest

```
func test_sendNewiMessage() {
    let app = XCUIApplication()
    app.launch()

    app.buttons[“new_message”].tap()

    let newMessage = app.staticTexts[“New Message”]
    let predicate = NSPredicate(format: “exists == true”)
    let expectation = XCTNSPredicateExpectation(predicate: predicate, object: newMessage)
    let result = XCTWaiter.wait(for: [expectation], timeout: 5)
    XCTAssertEqual(result, .completed)

    app.typeText(“iMessage Contact”)

    let newimessage = app.staticTexts[“New iMessage”]
    let newimessagePredicate = NSPredicate(format: “exists == true”)
    let newiMessageExpectation = XCTNSPredicateExpectation(predicate: newimessagePredicate, object: newimessage
    let newiMessageResult = XCTWaiter.wait(for:   [newiMessageExpectation], timeout: 5)
    XCTAssertEqual(newiMessageResult, .completed)

    let firstField = app.textFields[“messageField”]
```

```
    let result = XCTWaiter.wait(for: [expectation], timeout: 5)
    XCTAssertEqual(result, .completed)

    app.typeText("iMessage Contact")

    let newimessage = app.staticTexts["New iMessage"]
    let newimessagePredicate = NSPredicate(format: "exists == true")
    let newiMessageExpectation = XCTNSPredicateExpectation(predicate: newimessagePredicate, object: newimessage
    let newiMessageResult = XCTWaiter.wait(for:   [newiMessageExpectation], timeout: 5)
    XCTAssertEqual(newiMessageResult, .completed)

    let firstField = app.textFields["messageField"]

    firstField.typeText("test iMessage")
    app.buttons["send"].tap()

    let message = app.staticTexts["test iMessage"]
    let messagePredicate = NSPredicate(format: "exists == true")
    let messageExpectation = XCTNSPredicateExpectation(predicate: messagePredicate, object: message)
    let messageResult = XCTWaiter.wait(for: [messageExpectation], timeout: 5)
    XCTAssertEqual(messageResult, .completed)
}
```

```swift
import XCTest
class Robot {
    var app = XCUIApplication()

    func tap(_ element: XCUIElement, timeout: TimeInterval = 5) {
        let expectation = XCTNSPredicateExpectation(predicate: NSPredicate(format: “isHittable == true”), object: element)
        guard XCTWaiter.wait(for: [expectation], timeout: timeout) == .completed else {
            XCTAssert(false, “Element \(element.label) not hittable”)
        }
    }

    func assertExists(_ elements: XCUIElement…, timeout: TimeInterval = 5) {
        let expectation = XCTNSPredicateExpectation(predicate: NSPredicate(format: “exists == true”), object: elements)
        guard XCTWaiter.wait(for: [expectation], timeout: timeout) == .completed else {
            XCTAssert(false, “Element does not exist”)
        }
    }
}
```

Swift

```
class ConversationDetailRobot: Robot {

    private var messageType = “Message”
    lazy private var screenTitle = app.staticTexts[“New \(messageType)”]
    lazy private var contactField = app.textFields[“contact”]
    lazy private var cancel = app.buttons[“Cancel”]
    lazy private var messageField =     app.textFields[“messageField”]
    lazy private var sendButton = app.buttons[“send”]

    @discardableResult
    func checkScreen(messageType: String) -> Self {
        self.messageType = messageType
        assertExists(screenTitle, contactField, cancel, message   Field, sendButton)
        return self
    }

    @discardableResult
    func enterContact(contact: String) -> Self {
        tap(contactField)
        contactField.typeText(contact)
        return self
    }

    @discardableResult
    func enterMessage(message: String) -> Self {
        tap(messageField)
        messageField.typeText(message)
        return self
```

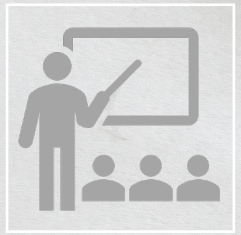
```swift
    @discardableResult
    func enterContact(contact: String) -> Self {
        tap(contactField)
        contactField.typeText(contact)
        return self
    }

    @discardableResult
    func enterMessage(message: String) -> Self {
        tap(messageField)
        messageField.typeText(message)
        return self
    }

    @discardableResult
    func sendMessage() -> Self {
        tap(sendButton)
        return self
    }

    @discardableResult
    func checkConversationContains(message: String) -> Self {
        let messageBubble = app.staticTexts[message]
        assertExists(messageBubble)
        return self
    }
}
```

Swift

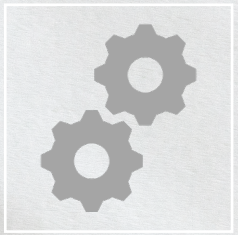
Robot Pattern

```
func test_sendNewiMessage() {

  let message = “test message”

   XCUIApplication().launch()

   ConversationListRobot()
     .newConversation()
     .checkConversationContains(message: “Message”)
     .enterContact(contact: “iMessage Contact”)
     .checkScreen(messageType: “iMessage”)
     .enterMessage(message: message)
      .sendMessage()
     .checkConversationContains(message: message)
  }
```

Agenda



Introduction



Robot Pattern

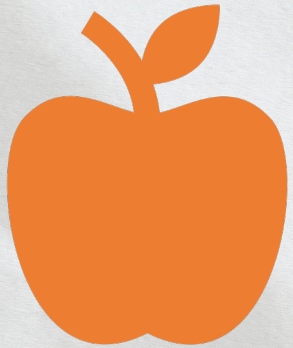
View Inspector

Snapshot Tests

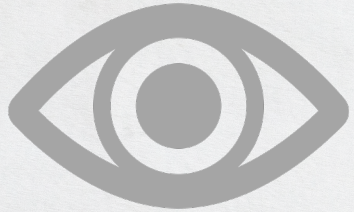
Agenda

View Inspector

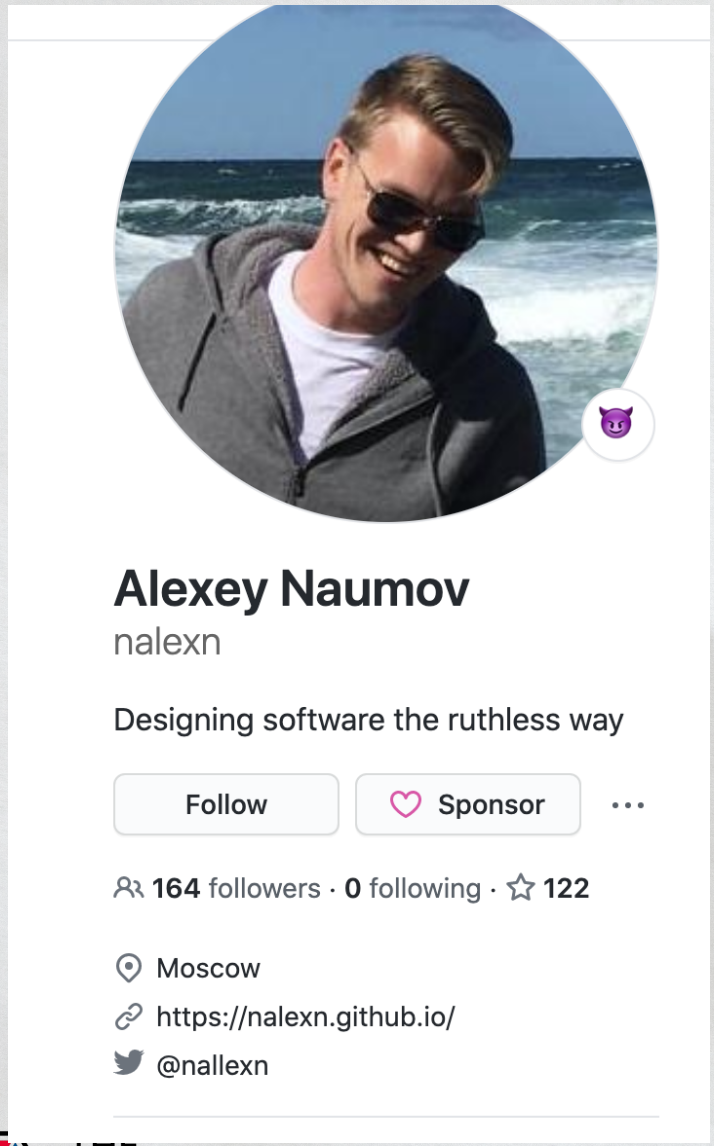
View Inspector



Apple Support



SwiftUI Views



- https://nalexn.github.io/swiftui-unit-testing/

Features

1. Verify the view's inner state

You can dig into the hierarchy and read the actual state values on any SwiftUI View:

```
func testVStackOfTexts() throws {
    let view = VStack {
        Text("1")
        Text("2")
        Text("3")
    }
    let text = try view.inspect().vStack().text(2).string()
    XCTAssertEqual(text, "3")
}
```

Features

2. Trigger side effects

You can simulate user interaction by programmatically triggering system-controls callbacks:

```
let button = try view.inspect().hStack().button(1)
try button.tap()

let list = try view.inspect().list()
try list[5].view(RowItemView.self).callOnAppear()
```

Features

3. Extract custom views from the hierarchy of any depth

It is possible to obtain a copy of your custom view with actual state and references from the hierarchy of any depth:

```
let sut = try view.inspect().tabView().navigationView()
    .overlay().anyView().view(CustomView.self).actualView()
XCTAssertTrue(sut.viewModel.isUserLoggedIn)
```

Features

- Views using **@Binding**
- Views using **@ObservedObject**
- Views using **@State**, **@Environment** or **@EnvironmentObject**
- ViewModifiers

Example

```
import XCTest
import ViewInspector
import SwiftUI
@testable import PocViewInspector

class PocViewInspectorTests: XCTestCase  {
```

Example

```
import XCTest
import ViewInspector
import SwiftUI
@testable import PocViewInspector

class PocViewInspectorTests: XCTestCase  {

    func testContentView_withText_shouldHaveHelloWorld() throws {
        let sut = ContentView()
        let text = try sut.body.inspect().vStack().text(0).string()
        XCTAssertEqual(text, "Hello, World!")
    }
```

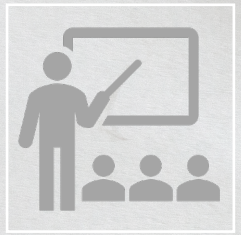
Example

```
import XCTest
import ViewInspector
import SwiftUI
@testable import PocViewInspector

class PocViewInspectorTests: XCTestCase  {

    func testContentView_withText_shouldHaveHelloWorld() throws {
        let sut = ContentView()
        let text = try sut.body.inspect().vStack().text(0).string()
        XCTAssertEqual(text, "Hello, World!")
    }

    func testContentView_withList_firstElementShouldBeHello0() throws {
        let sut = ContentView()
        let list = try sut.body.inspect().vStack().list(1)
        let firstText = try list.forEach(0).hStack(0).text(0).string()
        let secondText = try list.forEach(0).hStack(1).text(0).string()

        XCTAssertEqual(firstText, "Hello 0")
        XCTAssertEqual(secondText, "Hello 1")
    }
```

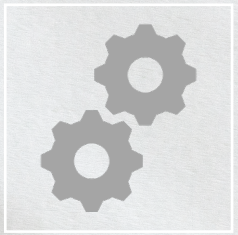
```swift
        let sut = ContentView()
        let text = try sut.body.inspect().vStack().text(0).string()
        XCTAssertEqual(text, "Hello, World!")
    }

    func testContentView_withList_firstElementShouldBeHello0() throws {
        let sut = ContentView()
        let list = try sut.body.inspect().vStack().list(1)
        let firstText = try list.forEach(0).hStack(0).text(0).string()
        let secondText = try list.forEach(0).hStack(1).text(0).string()

        XCTAssertEqual(firstText, "Hello 0")
        XCTAssertEqual(secondText, "Hello 1")
    }

    func testContentView_withCustomView_shouldHaveTextWithCustomText() throws {
        let sut = ContentView()

        let customView = try sut.body.inspect().vStack().view(CustomView.self, 2).actualView()
        let text = try customView.body.inspect().text().string()
        XCTAssertEqual(text, "Custom text")
    }

}
extension CustomView: Inspectable {}
```

Swift

Agenda



Introduction



Robot Pattern

View Inspector

Snapshot Tests

Agenda

Snapshot Tests

Snapshot Testing

Regression testing

UI Changes

Snapshot types

Features

- **Dozens of snapshot strategies**. Snapshot testing isn't just for `UIView`s and `CALayer`s. Write snapshots against *any* value.

Features

- **Dozens of snapshot strategies**. Snapshot testing isn't just for `UIView`s and `CALayer`s. Write snapshots against *any* value.
- **Write your own snapshot strategies**. If you can convert it to an image, string, data, or your own diffable format, you can snapshot test it! Build your own snapshot strategies from scratch or transform existing ones.

Features

- **Dozens of snapshot strategies**. Snapshot testing isn't just for `UIView`s and `CALayer`s. Write snapshots against *any* value.
- **Write your own snapshot strategies**. If you can convert it to an image, string, data, or your own diffable format, you can snapshot test it! Build your own snapshot strategies from scratch or transform existing ones.
- **No configuration required.** Don't fuss with scheme settings and environment variables. Snapshots are automatically saved alongside your tests.

Features

- **Dozens of snapshot strategies**. Snapshot testing isn't just for `UIView`s and `CALayer`s. Write snapshots against *any* value.
- **Write your own snapshot strategies**. If you can convert it to an image, string, data, or your own diffable format, you can snapshot test it! Build your own snapshot strategies from scratch or transform existing ones.
- **No configuration required.** Don't fuss with scheme settings and environment variables. Snapshots are automatically saved alongside your tests.
- **More hands-off.** New snapshots are recorded whether `isRecording` mode is `true` or not.

Features

- **Dozens of snapshot strategies**. Snapshot testing isn't just for `UIView`s and `CALayer`s. Write snapshots against *any* value.
- **Write your own snapshot strategies**. If you can convert it to an image, string, data, or your own diffable format, you can snapshot test it! Build your own snapshot strategies from scratch or transform existing ones.
- **No configuration required.** Don't fuss with scheme settings and environment variables. Snapshots are automatically saved alongside your tests.
- **More hands-off.** New snapshots are recorded whether `isRecording` mode is `true` or not.
- **Subclass-free.** Assert from any XCTest case or Quick spec.

Features

- **Dozens of snapshot strategies**. Snapshot testing isn't just for `UIView`s and `CALayer`s. Write snapshots against *any* value.
- **Write your own snapshot strategies**. If you can convert it to an image, string, data, or your own diffable format, you can snapshot test it! Build your own snapshot strategies from scratch or transform existing ones.
- **No configuration required.** Don't fuss with scheme settings and environment variables. Snapshots are automatically saved alongside your tests.
- **More hands-off.** New snapshots are recorded whether `isRecording` mode is `true` or not.
- **Subclass-free.** Assert from any XCTest case or Quick spec.
- **Device-agnostic snapshots.** Render views and view controllers for specific devices and trait collections from a single simulator.

Features

- **Dozens of snapshot strategies**. Snapshot testing isn't just for `UIView`s and `CALayer`s. Write snapshots against *any* value.
- **Write your own snapshot strategies**. If you can convert it to an image, string, data, or your own diffable format, you can snapshot test it! Build your own snapshot strategies from scratch or transform existing ones.
- **No configuration required.** Don't fuss with scheme settings and environment variables. Snapshots are automatically saved alongside your tests.
- **More hands-off.** New snapshots are recorded whether `isRecording` mode is `true` or not.
- **Subclass-free.** Assert from any XCTest case or Quick spec.
- **Device-agnostic snapshots.** Render views and view controllers for specific devices and trait collections from a single simulator.
- **First-class Xcode support.** Image differences are captured as XCTest attachments. Text differences are rendered in inline error messages.

Features

- **Dozens of snapshot strategies**. Snapshot testing isn't just for `UIView`s and `CALayer`s. Write snapshots against *any* value.
- **Write your own snapshot strategies**. If you can convert it to an image, string, data, or your own diffable format, you can snapshot test it! Build your own snapshot strategies from scratch or transform existing ones.
- **No configuration required.** Don't fuss with scheme settings and environment variables. Snapshots are automatically saved alongside your tests.
- **More hands-off.** New snapshots are recorded whether `isRecording` mode is `true` or not.
- **Subclass-free.** Assert from any XCTest case or Quick spec.
- **Device-agnostic snapshots.** Render views and view controllers for specific devices and trait collections from a single simulator.
- **First-class Xcode support.** Image differences are captured as XCTest attachments. Text differences are rendered in inline error messages.
- **Supports any platform that supports Swift.** Write snapshot tests for iOS, Linux, macOS, and tvOS.

Features

- **Dozens of snapshot strategies**. Snapshot testing isn't just for `UIView`s and `CALayer`s. Write snapshots against *any* value.
- **Write your own snapshot strategies**. If you can convert it to an image, string, data, or your own diffable format, you can snapshot test it! Build your own snapshot strategies from scratch or transform existing ones.
- **No configuration required.** Don't fuss with scheme settings and environment variables. Snapshots are automatically saved alongside your tests.
- **More hands-off.** New snapshots are recorded whether `isRecording` mode is `true` or not.
- **Subclass-free.** Assert from any XCTest case or Quick spec.
- **Device-agnostic snapshots.** Render views and view controllers for specific devices and trait collections from a single simulator.
- **First-class Xcode support.** Image differences are captured as XCTest attachments. Text differences are rendered in inline error messages.
- **Supports any platform that supports Swift.** Write snapshot tests for iOS, Linux, macOS, and tvOS.
- **SceneKit, SpriteKit, and WebKit support.** Most snapshot testing libraries don't support these view subclasses.

Features

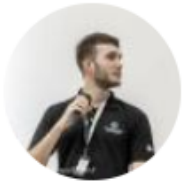
- **Dozens of snapshot strategies**. Snapshot testing isn't just for `UIView`s and `CALayer`s. Write snapshots against *any* value.
- **Write your own snapshot strategies**. If you can convert it to an image, string, data, or your own diffable format, you can snapshot test it! Build your own snapshot strategies from scratch or transform existing ones.
- **No configuration required.** Don't fuss with scheme settings and environment variables. Snapshots are automatically saved alongside your tests.
- **More hands-off.** New snapshots are recorded whether `isRecording` mode is `true` or not.
- **Subclass-free.** Assert from any XCTest case or Quick spec.
- **Device-agnostic snapshots.** Render views and view controllers for specific devices and trait collections from a single simulator.
- **First-class Xcode support.** Image differences are captured as XCTest attachments. Text differences are rendered in inline error messages.
- **Supports any platform that supports Swift.** Write snapshot tests for iOS, Linux, macOS, and tvOS.
- **SceneKit, SpriteKit, and WebKit support.** Most snapshot testing libraries don't support these view subclasses.
- **`Codable` support**. Snapshot encodable data structures into their JSON and property list representations.

Features

- **Dozens of snapshot strategies**. Snapshot testing isn't just for `UIView`s and `CALayer`s. Write snapshots against *any* value.
- **Write your own snapshot strategies**. If you can convert it to an image, string, data, or your own diffable format, you can snapshot test it! Build your own snapshot strategies from scratch or transform existing ones.
- **No configuration required.** Don't fuss with scheme settings and environment variables. Snapshots are automatically saved alongside your tests.
- **More hands-off.** New snapshots are recorded whether `isRecording` mode is `true` or not.
- **Subclass-free.** Assert from any XCTest case or Quick spec.
- **Device-agnostic snapshots.** Render views and view controllers for specific devices and trait collections from a single simulator.
- **First-class Xcode support.** Image differences are captured as XCTest attachments. Text differences are rendered in inline error messages.
- **Supports any platform that supports Swift.** Write snapshot tests for iOS, Linux, macOS, and tvOS.
- **SceneKit, SpriteKit, and WebKit support.** Most snapshot testing libraries don't support these view subclasses.
- **`Codable` support**. Snapshot encodable data structures into their JSON and property list representations.
- **Custom diff tool integration**.

```swift
import SnapshotTesting
import XCTest

class MyViewControllerTests: XCTestCase {
    func testMyViewController() {
        let contentView = SwiftUIView()
        let hostController = UIHostingController(rootView: contentView)
        assertSnapshot(matching: hostController, as: .image(on: .iPhoneSe))
        assertSnapshot(matching: hostController, as: .image(on: .iPhoneSe(.landscape)))
        assertSnapshot(matching: hostController, as: .image(on: .iPhoneX))
        assertSnapshot(matching: hostController, as: .image(on: .iPadMini(.portrait)))
    }
}
```

Swift



txaiwieser commented on 3 Jul

Also facing a lot of problems trying to snapshot SwiftUI views

👍 4

THE DEVELOPER'S CONFERENCE

ThoughtWorks®


Pros
Cons

Robot Pattern - Pros

- Reuse of code
- Clear tests
- Easy to create
- Accessibility

Robot Pattern - Cons

Visual specs

Mock

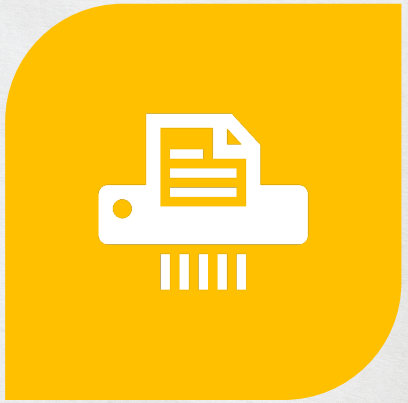
Production code

Slow

View Inspector - Pros

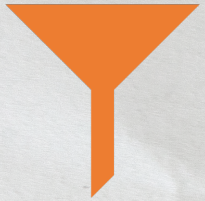
Full support to unit test
SwiftUI views and modifiers

Fast and easy to mock



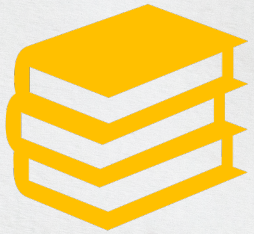
Open-source

View Inspector - Cons



Hard to find element to test in view hierarchy

New SwiftUI view types may not be supported



Library is maintained by the community

Snapshot - Pros

- Almost no efforts to update broken tests
- Trivial to write
- Verify different appearances, sizes and languages
- Obviously what has changed

Snapshot cons

BUGS

SIZE

FALSE NEGATIVES

COUPLING

Thank You