UI Testing for iOS applications

Sample project discussed here you can find at GitHub: https://github.com/alounce/SimpleTodo

Agenda

1. Configuring stubs for UI testing
2. UI tests classes
3. Recording UI tests
4. Tips and tricks for UI testing
5. How and in which scenarios UI tests can be useful for us
6. Useful resources

1. Configuring stubs for UI testing

In order to make tests stable and repeatable we have to stub all network communication.
At the same time we have security requirement to not include stubbed responses into app bundle
We have to pass stubs from outside and make that secure
The only way to pass data that I found was to use

```
XCUIApplication().launchArguments - which are [String]
XCUIApplication().launchEnvironment - which are [String: String]
```

In order to configure OHTTPStubs properly we need to pass several things

- HTTP method (GET, PUT, POST, DELETE)
- Path that we want to stub (for example: "api/controller/resource/3"
- HTTP status code that we expect to receive (200, 404, 500)
- Namely stub content

And all this we need to pass as Strings. We can came out with following solution:
1. Pass special argument within launchArguments which says that we are going to UI Testing mode and we would like to stub network layer calls
2. Pass stubs info within launchEnvironments dictionary where key and value will have format like

```
    Key -     "HTTPSTUB|GET|/api/todos"
    Value -   "201|{ encoded-json }"
```

In order to allow swift compiler to help us and get under control proper key-value format we can implement appropriate class, which encapsulates all this logic.
See `TestEnvironmentStubInfo.swift`
In result test code will look something like following:

UITest side

```
// Set flag that we are in UI testing mode
let app = XCUIApplication()
app.launchArguments.append(TestEnvironmentStubInfo.kUseHttpStubs)
// ADDING STUBS ---------------------------------------------------
// (C)reate
let newTodoJSON = ["id":24,"title":"Do it!","details":"some details","priority":
1,"category":"CHALLENGING","completed":false] as [String: AnyObject]
let createStub = TestEnvironmentStubInfo(method: .create, path: "/api/todos/", statusCode:
201, json: newTodoJSON)
app.launchEnvironment[createStub.key] = createStub.value
// (R)ead
let thisBundle = Bundle.init(for: SimpleTodoUITests.self)
if let stubInfo1 = TestEnvironmentStubInfo(method: .read, path: "/api/todos", statusCode:
200, fileName: "getTodos", bundle: thisBundle) {
  app.launchEnvironment[stubInfo1.key] = stubInfo1.value
}
// (U)pdate
...
```

And application side:

```
func application(_ application: UIApplication,
                         didFinishLaunchingWithOptions launchOptions:
[UIApplicationLaunchOptionsKey: Any]?) -> Bool {
  // Override point for customization after application launch.
  if ProcessInfo.processInfo.arguments.contains(TestEnvironmentStubInfo.kUseHttpStubs) {
    setupStubs(ProcessInfo.processInfo.environment)
  }
  return true
}
…
func setupStubs(_ environment: [String: String]) {
 print("🚚 SETUP STUBS")
 for (key, value) in environment {
  guard TestEnvironmentStubInfo.isStubInfo(key),
  let stubInfo = TestEnvironmentStubInfo(environmentKey: key, value: value) else
{ continue }
   stub(withTestEnvironmentInfo:stubInfo)
   print("STUB: \(stubInfo.description)")
 }
}
```

Now we are ready to proceed with UI tests.

2. UI tests classes

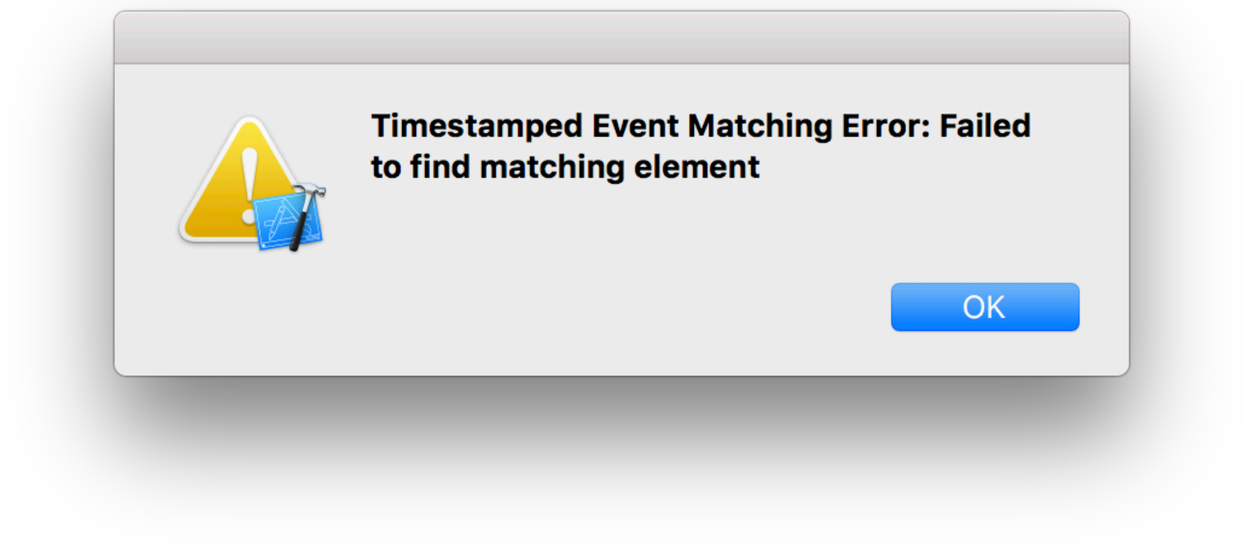
`XCUIApplication`
`XCUIElementQuery` (collection of elements)
`XCUIElement`

Show that at simple scenarios, how to query table cell by index, by identifier, buttons, tables,
Show that `XCUIElementQuery` is just a **Dynamic query** (think about it as about URL which brings data only when you access it)

3. Recording UI tests

- When you record tests - you cannot use stubs! So it will use real data - be aware!
- It not too stable, can crash the system, however not a big deal

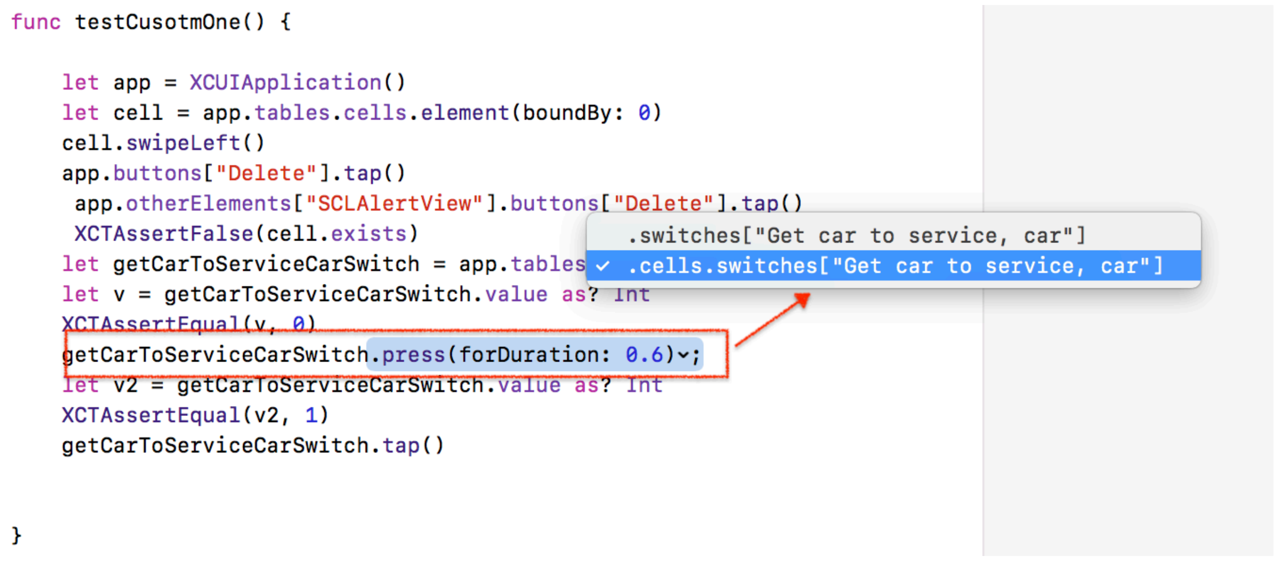
Can stop record with **Timestamped Event Matching Error: Failed to find matching element**



Timestamped Event Matching Error: Failed to find matching element

Usually it happens when you tap not at label/control. Just run recording again
What is important - it allows you to see accessibility identifiers. Also it helps to learn how to write UI tests
Or use Xcode->Open Developer Tools->**Accessibility Inspector**

- Mention about UI test recorded code tokens like



```
func testCusotmOne() {

    let app = XCUIApplication()
    let cell = app.tables.cells.element(boundBy: 0)
    cell.swipeLeft()
    app.buttons["Delete"].tap()
     app.otherElements["SCLAlertView"].buttons["Delete"].tap()
     XCTAssertFalse(cell.exists)
    let getCarToServiceCarSwitch = app.tables
    let v = getCarToServiceCarSwitch.value as? Int
    XCTAssertEqual(v, 0)
    getCarToServiceCarSwitch.press(forDuration: 0.6);
    let v2 = getCarToServiceCarSwitch.value as? Int
    XCTAssertEqual(v2, 1)
    getCarToServiceCarSwitch.tap()


}
```

```
.switches["Get car to service, car"]
✓ .cells.switches["Get car to service, car"]
```

4. Tips and tricks for UI testing

- Remember that main reason of UI testing is to test certain scenario
- UI tests executes several times slower than Unit tests, be aware of that
- Use recording mode to quickly get identifiers
- Use Accessibility Inspector for the same purpose
- Remember that queries in UI testing resolves dynamically, when flow access them
- We can flexible configure which tests we need to run during testing (current schema, test configuration)
- We can tweak UI test report (XCTContext.runActivity)
- We can make screenshots of the application automatically (See in code)

5 How and in which scenarios UI tests can be useful for us

- Let's imagine that QA will do that scenario manually for **2 minutes.**
- Then for running such scenarios we spent 2 minutes * 18 builds = **36 minutes**
- Scenario can be applied to different devices (iPhones 5S, SE, 6/S, 6+/S, 8/8+, X, iPads 5Gen, Air/2, Mini 2, Pros 9.7/10.5/11.9) about 17 different devices . So 36 min * 17 devices = **612 minutes (or more than 10 hours)**
- Usually we test for 2 OS (now it is 10/11) - 10 hours * 2 = **20 hours**
- Let's imagine we are delivering 12th sprint so for all this time for testing such scenario we will spent 12 * 20 = **240 hours!**

For writing this test I spent about **3h**, and I still was at learning curve. Expect that it can be even quicker

3 <<< 240

And finally, we can run several tests in parallel from Terminal

```
xcodebuild test -workspace SimpleTodo.xcworkspace -scheme
SimpleTodo -sdk iphonesimulator -destination 'platform=iOS
Simulator,name=iPhone X' -destination 'platform=iOS
Simulator,name=iPad Air 2' -destination 'platform=iOS
Simulator,name=iPhone SE'
```

6 Useful resources

There are some useful resources devoted to UI testing:

- UI Testing Cheat Sheet and Examples (http://masilotti.com/ui-testing-cheat-sheet/)
- UI Testing in Xcode WWDC-2015 (https://developer.apple.com/videos/play/wwdc2015/406/)
- What's New in Testing WWDC-2017 (https://developer.apple.com/videos/play/wwdc2017/409/)