ROBOT FRAMEWORK 🤖

UITest 101

Cronograma

- UITest 101
- ROBOT FRAMEWORK
- Minha abordagem de ROBOT

UITest 101

Testes de tela auxiliam a checagem dos elementos da tela e fluxos esperados como foram desenvolvidos. Um grande diferencial para os testes unitários, é que no caso do Teste de tela nós precisamos necessariamente do aplicativo rodando no nosso host para executar os testes, onde estaremos checando o estado da aplicação como um todo.

UITest 101 - porque fazer teste de tela?

O teste de tela vai auxiliar em:

- previnir o teste em feature antigas
- garantir casos de uso integros
- performance da tela
- checar localização
- snapshot para App Store

UITest 101 - trade-offs

Fazer testes de tela sempre é de grande valia para um projeto, mas é sempre bom lembrar seus *trade-offs*

- levam mais tempo para desenvolver que os unitários
- demoram mais tempo para rodar que os unitários
- mudanças visuais nos apps, além de comuns, causam impacto direto nesses testes.

UITest 101 - XCUIApplication

Esta é a representação do aplicativo na tela do *host*, e é com ele que buscaremos por elementos na tela, além de garantir o

```
XCUIApplication().launch()
```

o `launch()` é sincrono, e garante que ao voltar já é possivel tratar eventos na aplicação.

UITest 101 - record

O Xcode nos da uma ferramenta muito poderosa para fazer testes de tela, com o `record` podemos gravar as nossa ações na tela. Ele é muito bom para criar de forma quase que automática um fluxo de ações esperados na tela.

Porém, como o código é autogerado, na maioria dos casos precisa ser revisado para melhorar sua leitura, ações ou até mesmo retirar passos desnecessários.

O `record` grava as ações, mas não faz ***asserts***, então cabe ao dev incrementar o teste com os ***asserts***

Demo

- criar nosso primeiro teste de UI
  - **record**
  - **asserts**

```
#Build Test Target:

⇧⌘U (shift + command + U)
```

UITest 101 - XCUIElementQuery

Tendo em mãos o `XCUIApplication` podemos procurar por elementos na tela, e esses podem ser dos mais diversos tipos, e para cada tipo de elemento temos uma ação de query específica, dentre elas:

- buttons
- staticTexts
- tableViews
- collectionViews
- images
- etc... (`XCUIElementTypeQueryProvider`)

UITest 101 - XCUIElement

Buscamos pelo nosso elemento, agora é hora de aplicar ações sobre ele, e pra isso um `XCUIElement` tem diversas ações:

- tap()
- doubleTap()
- swipes
- drag

Demo

- vamos buscar por um elemento especifico da tela (***accessibilityIdentifier***)

ROBOT FRAMEWORK

ROBOT FRAMEWORK

> Robot Framework is a generic open source automation framework for acceptance testing, acceptance test driven development (ATDD), and robotic process automation (RPA). It has simple plain text syntax and it can be extended easily with libraries implemented using Python or Java.
>
> ...
>
> Robot Framework itself is open source software released under Apache License 2.0, and most of the libraries and tools in the ecosystem are also open source. The framework was initially developed at Nokia Networks and was open sourced in 2008.
>
> > *https://robotframework.org/*

Exemplo

```
*** Settings ***
Documentation     A test suite with a single test for valid login.
...
...               This test has a workflow that is created using keywords in
...               the imported resource file.
Resource          resource.robot

*** Test Cases ***
Valid Login
    Open Browser To Login Page
    Input Username    demo
    Input Password    mode
    Submit Credentials
    Welcome Page Should Be Open
    [Teardown]    Close Browser
```

ROBOT FRAMEWORK

A princípio o framework é para testes e automatizações de aplicações WEB, que nos dá um monte de `keywords`, onde cada uma dessas `keywords` são fáceis de ler/escrever e são capazes de executar uma ação epecifica, dando a capacidade de criar testes com pouca ou quase nenhuma ação no código. Você pode criar as suas próprias `keywords` para reuso e complementação das já existentes.

É possivel estruturar os casos de teste com `Gherkin`, e também pode ser utilizado para automatizar processos repetitivos, por exemplo: gerar um relatório mensal de gastos e enviar para os gestores.

Sua modularidade o torna capaz de integrar com diversas bibliotecas, como Selenium e Appium

Minha abordagem de ROBOT

Minha abordagem de ROBOT

Não vou mostrar como o Appium funciona (seria outra talk só disso...)

Quero mostrar como as premissas do ROBOT FRAMEWORK podem nos ajudar a criar testes de tela mais facilmente, de fácil manutenção e leitura.

Minha abordagem de ROBOT

Principios:

- Keywords para facilitar o uso das ações do usuário com a aplicação
- dividir os casos de uso por Contexto
- identificar com qual tela estamos trabalhando

UIBOT

POD de UITest baseado no ROBOT FRAMEWORK (feito por mim... rsrs)

Esse POD te da algumas funções prontas para triggar ações e fazer asserts de elementos do UIKit

UIBOT - Conceito

Cada BOT representa uma TELA, e essa deve conter todas as interações possíveis, além dos asserts necessários. Uma boa prática é sempre buscar elementos pelo ***accessibilityIdentifier***.

Outro elemento muito importante é o `trait`, esse elemento define a tela e assim que instanciado um BOT ele vai buscar pelo `trait` para garantir que as ações serão feitas na TELA esperada.

Demo

- criar BOT com UIBOT
- criar um Test utilizando BOTs

Recapitulando - UITest

- Testes de tela dependem da aplicação aberta no device
- `record` auxilia para criação de fluxos de testes
  - o código autogerado precisa de revisão
  - deve ser adicionado *asserts* no código
- XCUIElementQuery - para buscar elementos na tela
- XCUIElement - O elemento específico da tela, que pode receber ações, como:
  - tap
  - swipes
  - etc..

Recapitulando - ROBOT

- Padrão de desenvolvimento de testes de Tela
- disponibiliza `keywords` para facilitar a criação dos teste
- além de testar ele pode ser utilizado para automatizar processos do sistema (RPA)
- sua modularidade permite integração com várias bibliotecas

Contato

tulio.bazan@dextra-sw.com