Automatizando testes sem sofrimento

pagseguro UOL

Rodrigo Amaro

Especialista Android no PagSeguro

Graduado em Engenharia da Computação

Pós-Graduado em Análise de Sistemas

pagseguro UOL

Sobre o PagSeguro

- Pioneiro e Líder no mercado brasileiro de meios de pagamento online
  - Mais de 600 mil lojas cadastradas
  - Mais de 40 milhões de compradores usam o PagSeguro
- Aplicativo PagSeguro lançado em Abril de 2013
  - Mais de 2 milhões de downloads (Android)
  - Presente nas plataformas Android, iOS e Windows Phone
  - Uso de tecnologias como Audio Jack Data Transmission e Bluetooth

pagseguro UOL

Agenda

- Testes de software
  - Automação de testes
  - Motivadores
- Histórico no PagSeguro
- Aprendendo por exemplo
  - Implementação básica de testes locais
  - Evolução da arquitetura de testes
  - Expandindo para testes End-to-End
- Considerações finais

pagseguro UOL

Dimensões de qualidade

Segundo Philippe Kruchten[1], existem pelo menos três dimensões de qualidade que precisam ser consideradas antes de se iniciar um ciclo de testes

- Confiança
- Funcionalidade
- Performance

[1] The rational unified process: An introduction (boston, addison-Wesley, 1999)

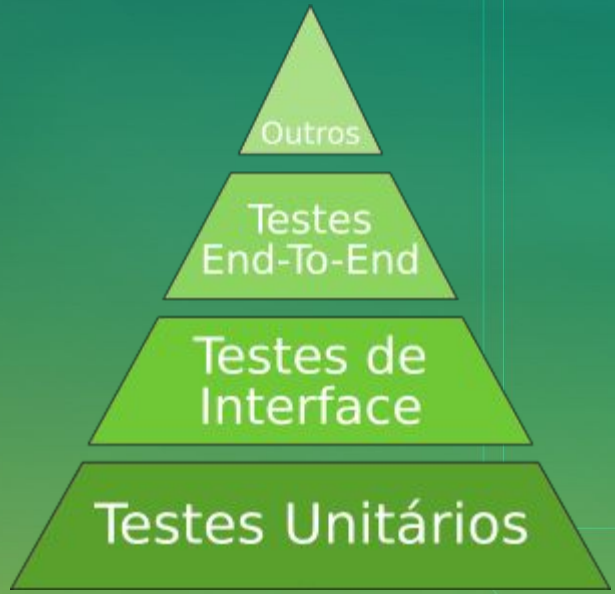
Testes automatizados

Permitem:

- Definir uma linha processual que pode ser executada inúmeras vezes para avaliar diversas características da aplicação
- Agendar a execução de testes com base em regras ou eventos
- Documentar as funcionalidades implementadas de modo que elas possam ser validadas

Motivadores

- Garantir o nível de qualidade das entregas
- Rápida evolução do produto
- Permitem avaliar diversos cenários e condições
- Maior liberdade aos QA’s para testes exploratórios



pagseguro UOL

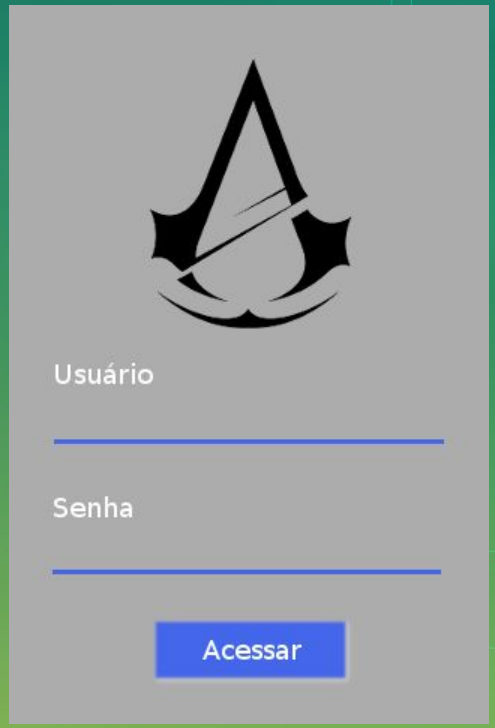
Histórico no PagSeguro

- Calabash
  - Baseado no Cucumber
  - Código em Ruby
  - Possibilidade de reutilização para iOS
- Robotium
  - Código em Java
  - Maior facilidade para trabalhar com Android
- Espresso
  - Comandos mais legíveis
  - Fornecido pelo Google

pagseguro UOL

Caso de Uso - Tela de login

- Usuário deve fornecer email e senha;
- Se o par (email e senha) estiver incorreto, sinalizar ao usuário
- Se estiverem corretos uma mensagem de sucesso deve ser exibida
- Havendo uma falha de rede oferecer a opção de tentar novamente



pagseguro UOL

Usando o Repositório

- Para facilitar o acompanhamento o código desta apresentação está disponível no seguinte repositório:

  https://bitbucket.org/rod_a_r/testes-automatizados.git

- O campo amarelo no canto superior representa a TAG com o código referente ao momento atual da apresentação

Configurando o Espresso

Vamos configurar nossa aplicação seguindo as recomendações da documentação do Espresso disponibilizada em:
https://google.github.io/android-testing-support-library/docs/espresso/setup/index.html

- Desabilitar as animações no dispositivo;
- Adicionar as dependências do Espresso no arquivo build.gradle do módulo;
- Configurar o Instrumentation Runner no mesmo arquivo.

Configurando o Espresso

Error:Conflict with dependency 'com.android.support:support-annotations'. Resolved versions for app (24.1.1) and test app (23.1.1) differ. See http://g.co/androidstudio/app-test-app-conflict for details.

Existem algumas soluções para o problema:

- Excluir o módulo defeituoso (support-annotations) das bibliotecas que causaram o conflito (espresso-core e rules)
- Forçar a versão a ser utilizada para garantir compatibilidade

http://stackoverflow.com/questions/28999124/resolved-versions-for-app-22-0-0-and-test-app-21-0-3-differ

Escrevendo os primeiros testes

Entendendo o funcionamento do Espresso:

- Como escrever o teste?
- Verificando valores, visibilidade e comportamento
- Efetuando ações
- Rodando os primeiros testes

https://google.github.io/android-testing-support-library/docs/espresso/setup/index.html#add-the-first-test

Tratando diferentes tamanhos de tela

- Qual o foco do seu teste?
  - Layout?
  - Experiência do usuário?
  - Funcionalidade?
- O que é mais importante para seu produto?
  - Aparecer o logo da sua empresa?
  - Garantir o acesso do usuário?

Tratando diferentes tamanhos de tela

- Ferramentas para resolver
  - isDisplayed()
  - doesNotExist()
  - scrollTo()

Testando múltiplos cenários

- Usar servidor de QA para realizar chamadas
- Resposta Hardcoded
- Mocks
- Serviço Local

Simulando a entrada de dados

Quando teclado vira um problema...

- Entrada de valores durante os testes
- typeText()
- closeSoftwareKeyboard()
- windowSoftInputMode

https://developer.android.com/guide/topics/manifest/activity-element.html#wsoft

Espresso - Resumo

- Espresso permite:
  - Verificar a presença de elemento baseando-se em regras (matchers)
  - Efetuar ações em elementos encontrados através de regras
- Problemas:
  - Conflito de versões entre bibliotecas de teste e de uso do app
  - Falhas do mecanismo de teste podem impedir a execução dos mesmos
- Dificuldades:
  - Simular diversos cenários sem impactar na execução do app em produção
  - Os diversos tamanhos de dispositivos devem ser tratados conforme a necessidade
  - Garantir um ambiente estável com pouco código

Usando mocks

- São objetos que simulam o comportamento de objetos reais
- Permitem fornecer contexto em testes automatizados
- Facilitam a escrita do teste

Literatura:

- https://www.amazon.com.br/Growing-Object-Oriented-Software-Guided-Tests/dp/0321503627
- https://www.amazon.com/gp/product/B00QS2HXUO/

pagseguro UOL

Configurando o Mockito

Etapas:

- Adicionando a dependência no build.gradle
- Revisando o código para injeção dos mocks
- Alterando o comportamento de início da Activity durante o teste
- Alterando o primeiro teste

Links:

http://mockito.org/

Configurando o Mockito

Problemas com a geração de código:

- Originalmente o Mockito gera .class
- Para que o Android possa reconhecer as classes geradas elas devem ser compatíveis com a Dalvik VM (.dex)
- Para isso usamos uma biblioteca chamada DEXMAKER

Links:

https://github.com/crittercism/dexmaker

Utilizando o Mockito

- Criando um objeto “mockado” com mock(class)
- Configurando seu mock
  - doReturn… when…
  - doThrow… when…
  - doAnswer… when...

Mockito - Resumo

- Usando mocks você consegue:
  - Evitar implementações de interface desnecessárias
  - Fornecer contexto para diversos cenários de teste
  - Facilitar a escrita dos testes
  - Evitar a necessidade de código Hard Coded em arquivos de produção
- Problemas e dificuldades:
  - Conversão das classes geradas pelo Mockito para DEX
  - Necessidade de iniciar manualmente a Activity para testes
  - Eventualmente surgem alguns problemas referentes aos código gerado pelo DEX

Injeção de dependências

- Baseado no conceito de Inversão de Controle (IoC) onde a classe foco não deve se preocupar em criar instâncias de suas dependências
- Temos uma figura central cuja responsabilidade é fornecer das dependências requeridas por outras classes
- Permite diminuir o acoplamento entre classes
- Facilita a escrita de testes

Links:

http://martinfowler.com/articles/injection.html
http://www.vogella.com/tutorials/Dagger/article.html

pagseguro UOL

Configurando o Dagger

- Declarando as dependências
- Configurando o processador de anotações
- Declarando módulos, componentes e dependências
- Método inject

Links:

http://google.github.io/dagger/
https://bitbucket.org/hvisser/android-apt

Dagger - Resumo

- Mais boilerplate code
- Melhor controle das dependências e configurações para testes
- Dificulta o envio de código de testes para produção

Testes End-To-End: Rx, Retrofit e SoapUI

- Utilização do Rx para facilitar o tratamento de chamadas ao serviço de Login em uma thread separada
- Retrofit para realizar as chamadas ao serviço de Login
- Uso do Soap UI para simular o serviço de Login

Links:

https://github.com/ReactiveX/RxJava
http://square.github.io/retrofit/
https://www.soapui.org/

Configurandos os servicos no Soap UI

- Criando um projeto de serviço
- Criando um endpoint
- Criando as respostas do EndPoint
- Definindo as respostas com base no Request usando Groove

Escrevendo testes End-to-End

- Configuração do ambiente para uso do serviço
- Implementação da camada de consumo do serviço
- Sincronismo da execução do teste
- IdlingResource

Categorizando testes

- Por que categorizar?
- Que tipo de categorias devo criar?
- Como categorizar?
- Executando categorias de testes

Dicas de última hora

- Desligue as animações durante a execução dos testes para não confundir o Espresso
- Caso comece a receber erros do tipo “No tests found” tente um Clean Project ou reinicie o Android Studio
- Procure categorizar os tipos de Teste de modo a agrupar pela característica do teste.
- Utilize nomenclaturas de fácil entendimento para ter clareza do que o teste realiza

Estude, pesquise, pergunte...

“I have not failed. I've just found 10,000 ways that won't work.”

Thomas A. Edison

pagseguro UOL

Dúvidas?

Obrigado

https://plus.google.com/+RodrigoAmaroRodrigues

https://br.linkedin.com/in/rodrigoamarorodrigues

pagseguro UOL