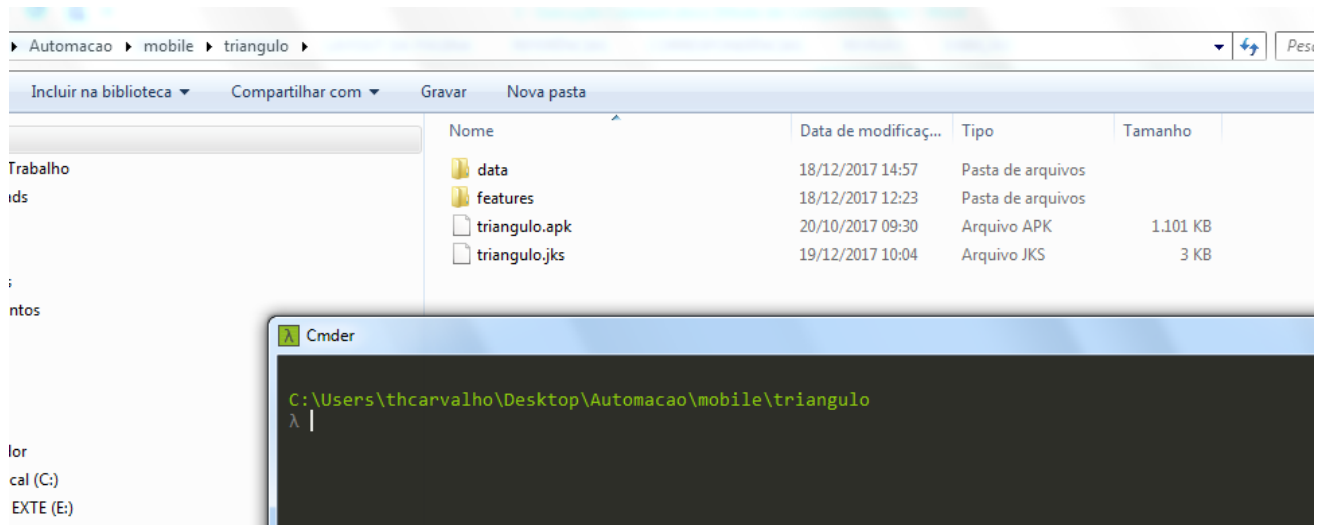
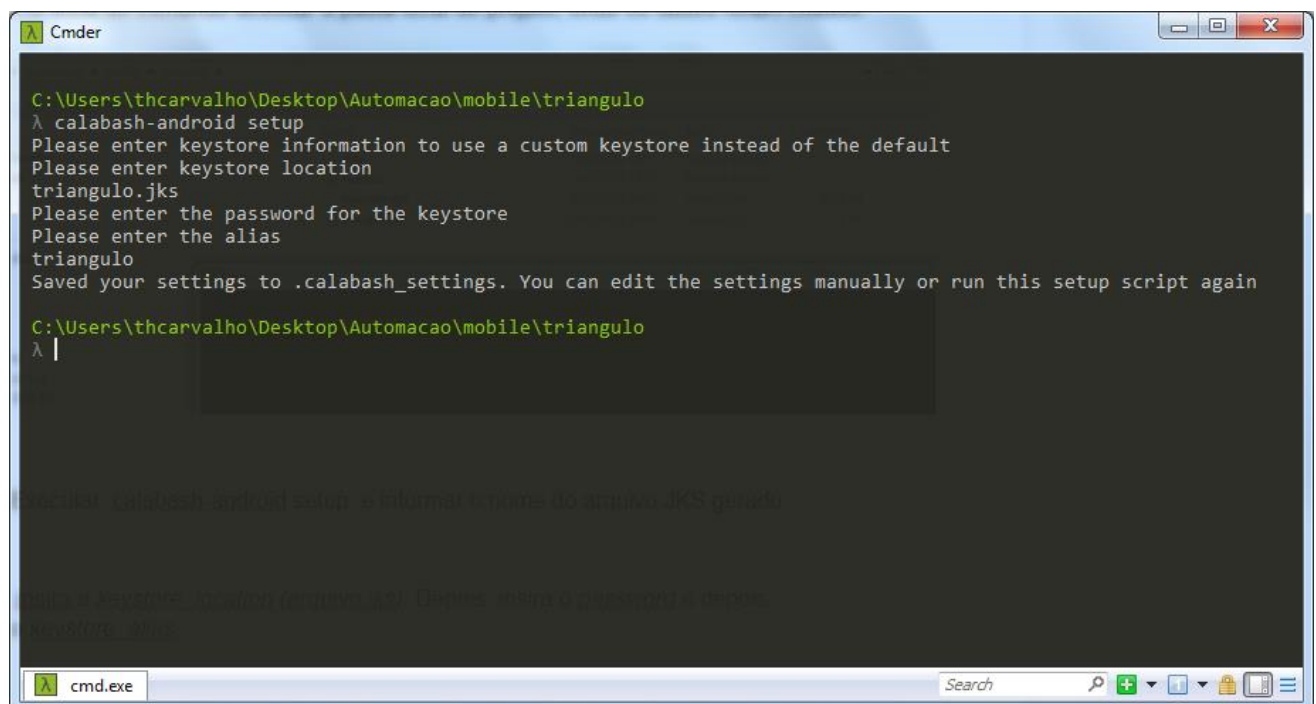


1 – Validando assinatura do APK

Via linha de comando acessar a pasta local do projeto, onde os dados foram criados



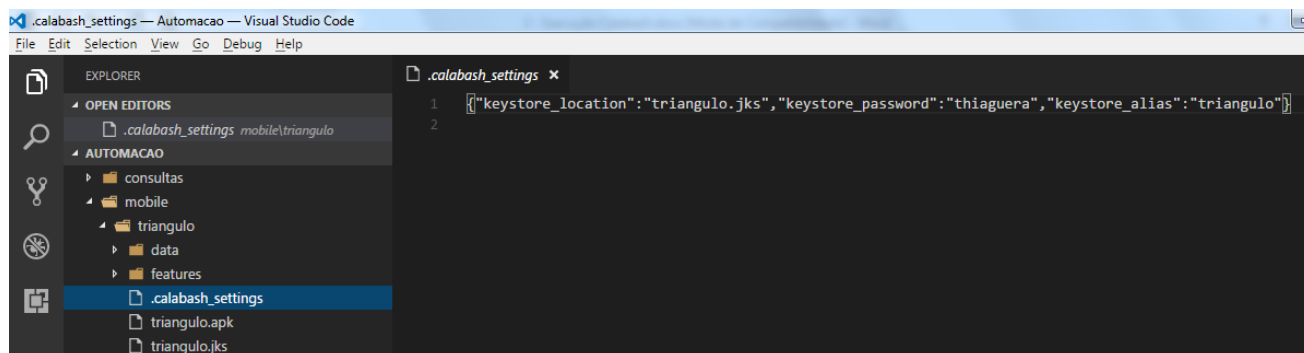
Executar: calabash-android setup, e informar o nome do arquivo JKS gerado. Depois, insira o *password* em seguida o *alias*.



Será criado na pasta do projeto, o arquivo calabash_settings

Nome	Data de modificaç...	Tipo	Tamanho
data	18/12/2017 14:57	Pasta de arquivos	
features	18/12/2017 12:23	Pasta de arquivos	
.calabash_settings	19/12/2017 12:19	Arquivo CALABAS...	1 KB
triangulo.apk	20/10/2017 09:30	Arquivo APK	1.101 KB
triangulo.jks	19/12/2017 10:04	Arquivo JKS	3 KB

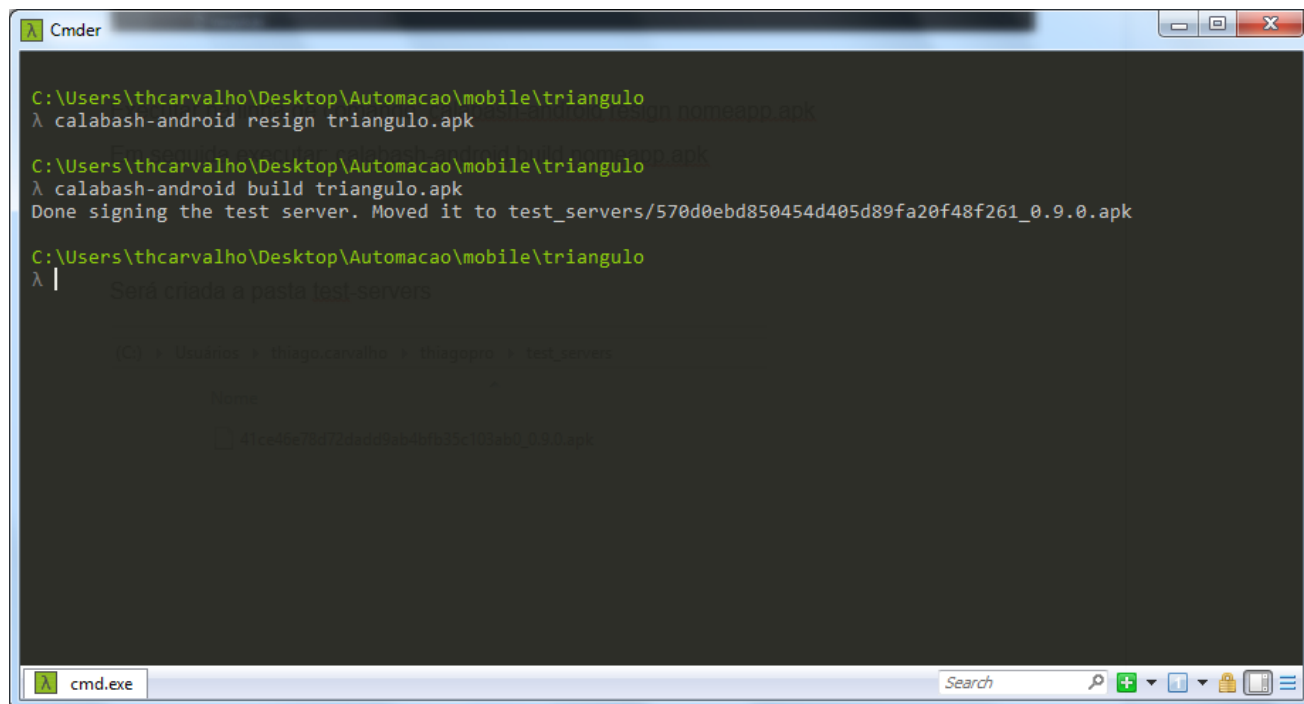
Depois de gerado, as informações podem ser alteradas manualmente pelo editor de texto



2 – Iniciando o servidor de testes

Executar na linha de comando: calabash-android resign nomeapp.apk

Em seguida executar: calabash-android build nomeapp.apk

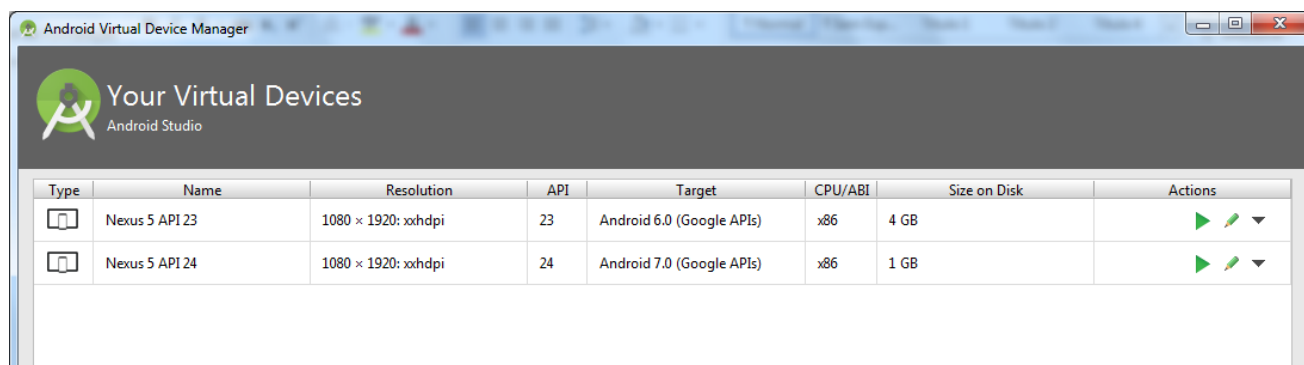


Será criada a pasta test-servers, dentro dela foi criado o servidor de testes

Nome	Data de modificaç...	Tipo	Tamanho
data	18/12/2017 14:57	Pasta de arquivos	
features	18/12/2017 12:23	Pasta de arquivos	
test_servers	19/12/2017 14:45	Pasta de arquivos	
.calabash_settings	19/12/2017 12:19	Arquivo CALABAS...	1 KB
triangulo.apk	19/12/2017 14:44	Arquivo APK	1.101 KB
triangulo.jks	19/12/2017 10:04	Arquivo JKS	3 KB

3 – Iniciar apk no emulador

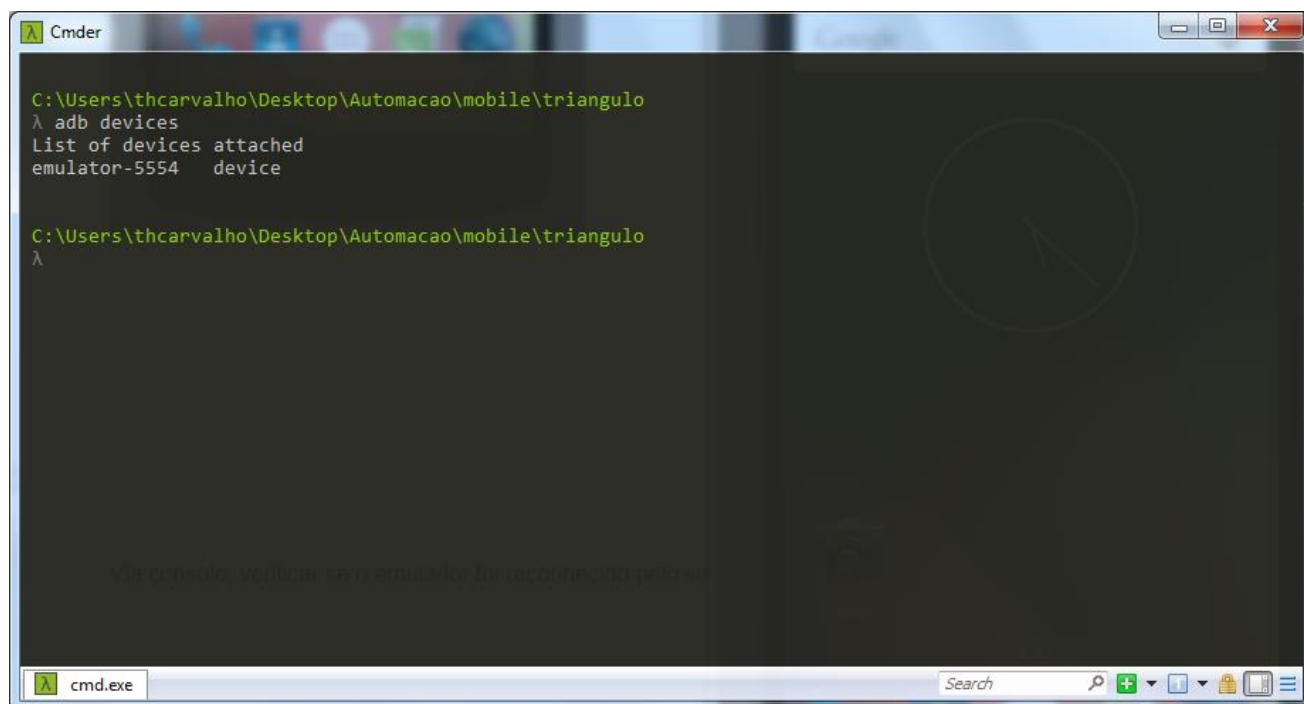
Selecionar uma versão configurada do android SDK, abrir o emulador



https://medium.com/@thi_carva

Via console, verificar se o emulador foi reconhecido pelo sistema, executar: adb devices

O console vai listar todos dispositivos disponíveis para teste



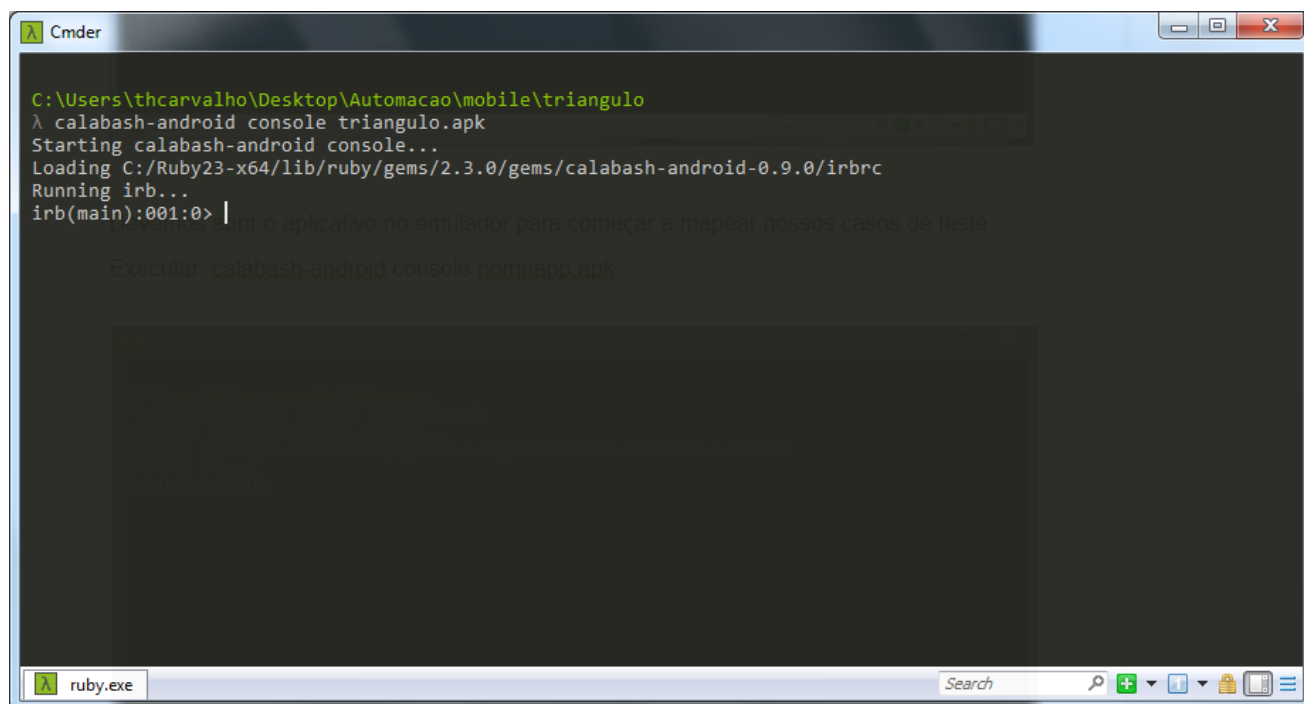
```
C:\Users\thcarvalho\Desktop\Automacao\mobile\triangulo
λ adb devices
List of devices attached
emulator-5554    device

C:\Users\thcarvalho\Desktop\Automacao\mobile\triangulo
λ
```

Devemos abrir o aplicativo no emulador para começar a mapear nossos casos de teste

Executar: calabash-android console nomeapp.apk

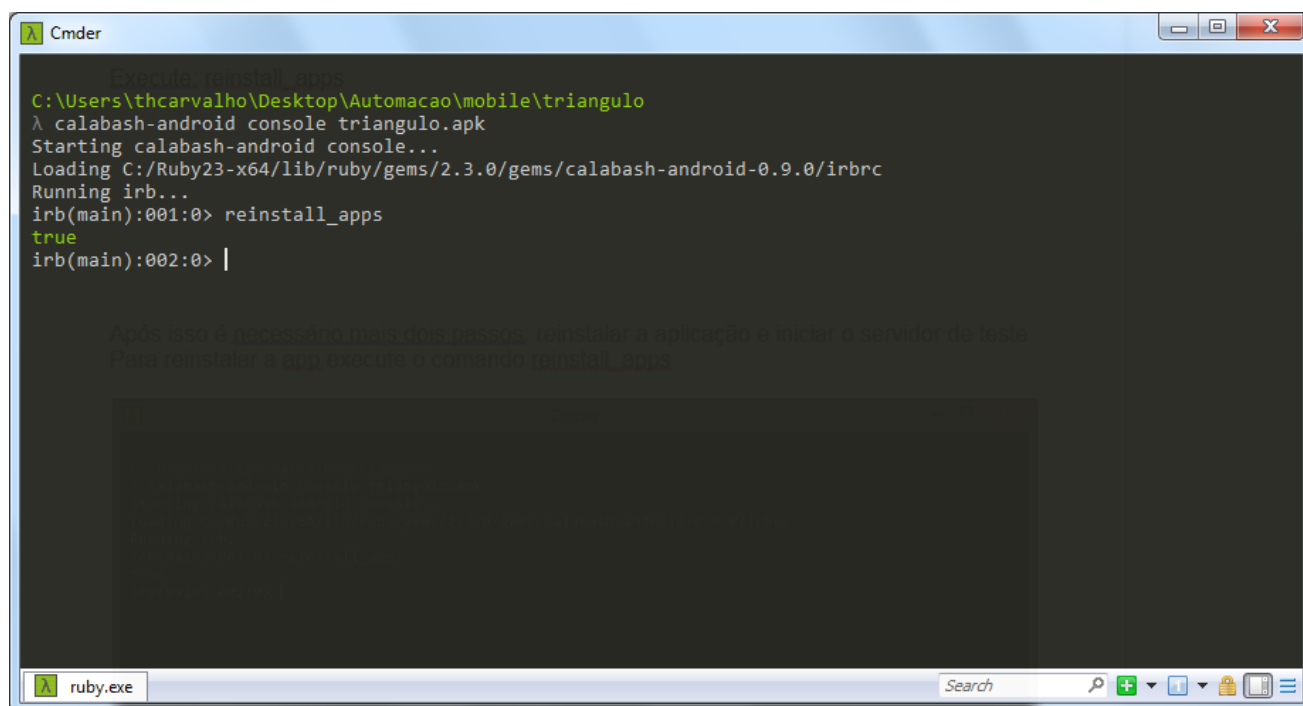
O console inicia a interação do Ruby, que vai conversar com o servidor



```
C:\Users\thcarvalho\Desktop\Automacao\mobile\triangulo
λ calabash-android console triangulo.apk
Starting calabash-android console...
Loading C:/Ruby23-x64/lib/ruby/gems/2.3.0/gems/calabash-android-0.9.0/irbrc
Running irb...
irb(main):001:0> |
```

Executar: `reinstall_apps`

O sistema irá reinstalar a aplicação e iniciará o servidor de teste

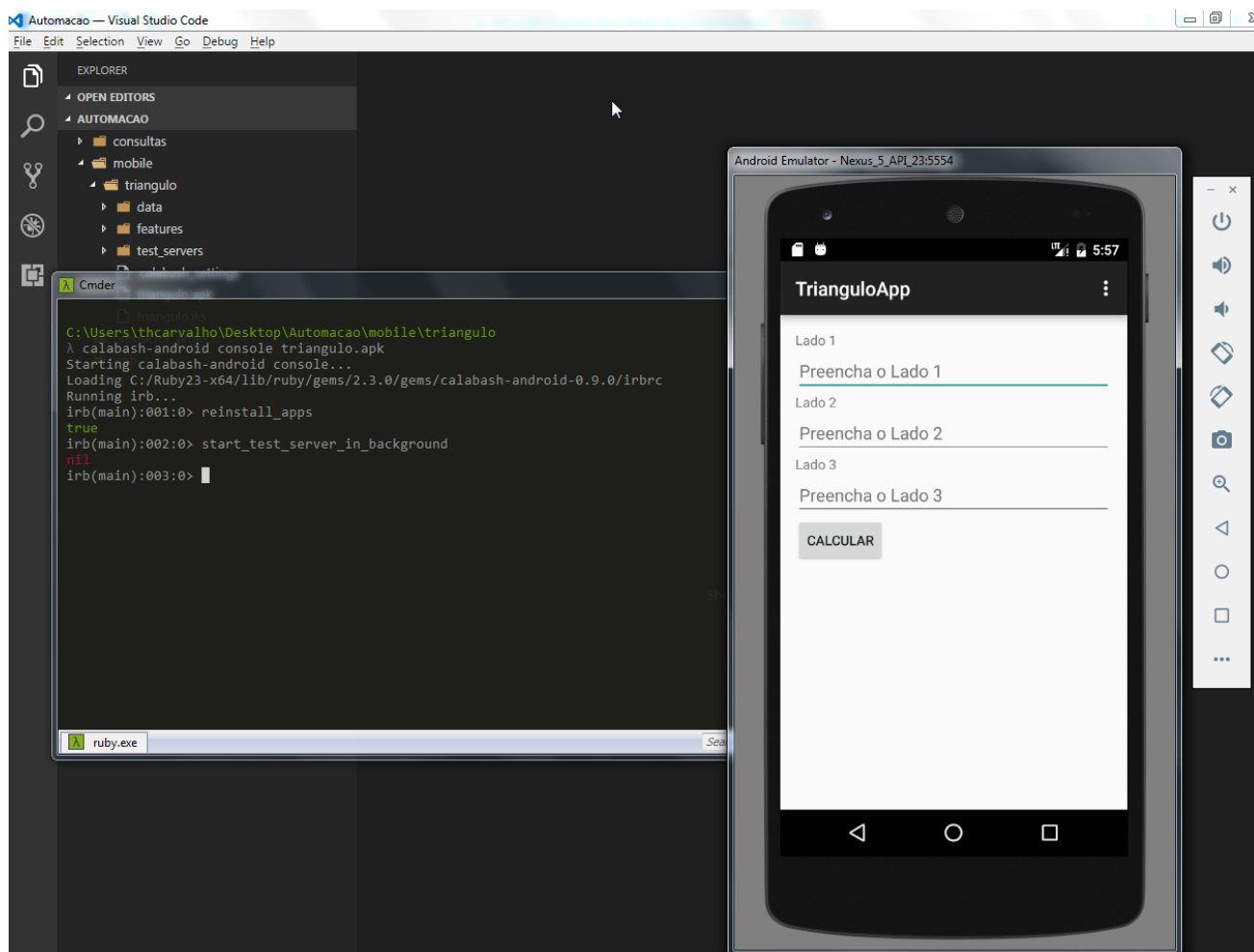


```
C:\Users\thcarvalho\Desktop\Automacao\mobile\triangulo
λ calabash-android console triangulo.apk
Starting calabash-android console...
Loading C:/Ruby23-x64/lib/ruby/gems/2.3.0/gems/calabash-android-0.9.0/irbrc
Running irb...
irb(main):001:0> reinstall_apps
true
irb(main):002:0> |
```

Após isso é necessário mais dois passos: reinstalar a aplicação e iniciar o servidor de teste
Para reinstalar a app execute o comando `reinstall_apps`

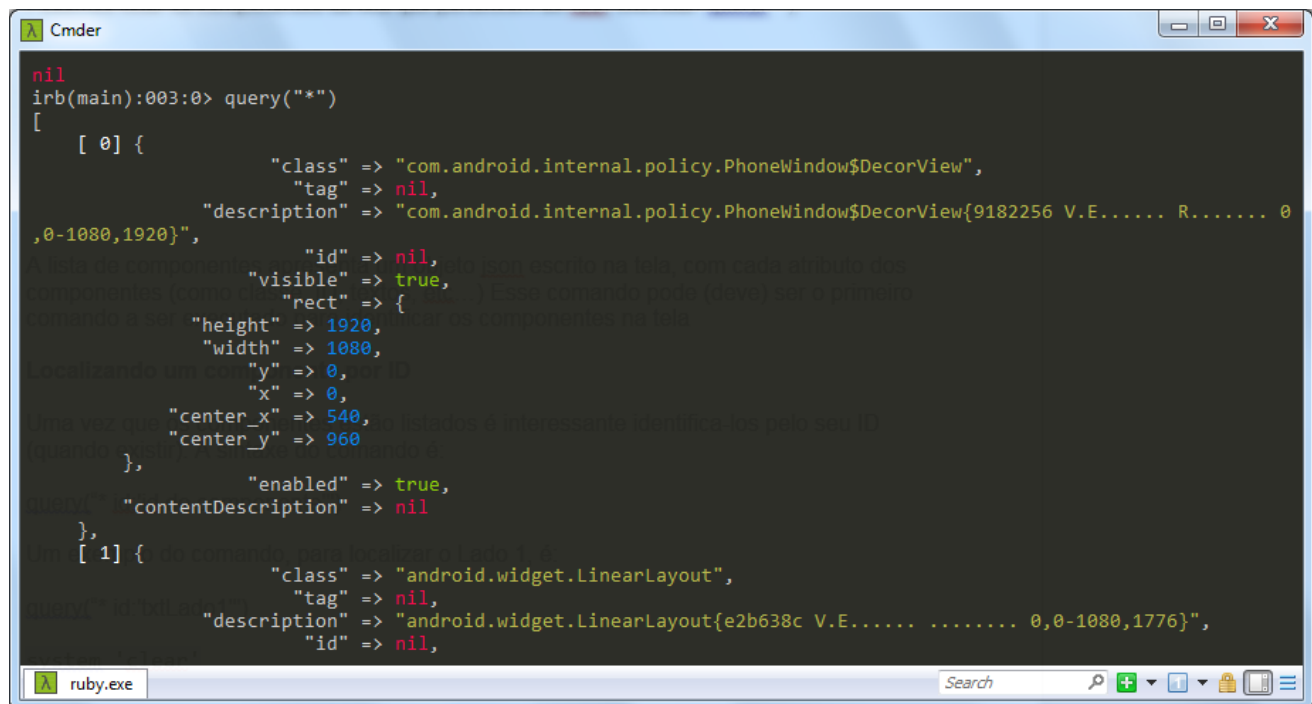
Executar: `start_test_server_in_background`

Após reinstalar o app, o aplicativo é inicializado no emulador para localizar os componentes



4 – Listando os componentes da tela

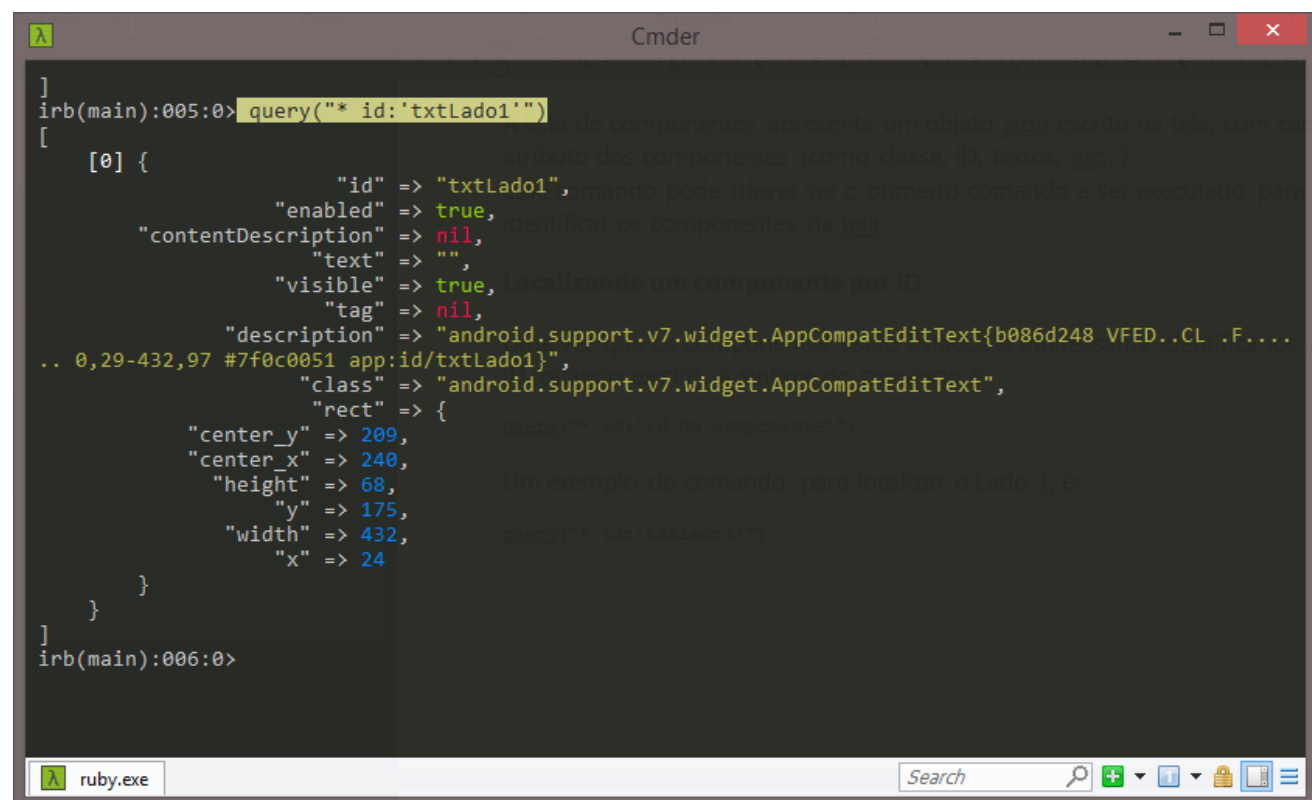
Podemos listar os componentes da tela que pertencem ao app, executar: `query("*")`



```
irb(main):003:0> query("*")
[
  [ 0] {
    "class" => "com.android.internal.policy.PhoneWindow$DecorView",
    "tag" => nil,
    "description" => "com.android.internal.policy.PhoneWindow$DecorView{9182256 V.E..... R..... 0
,0-1080,1920}",
    "id" => nil,
    "visible" => true,
    "rect" => {
      "height" => 1920,
      "width" => 1080,
      "y" => 0,
      "x" => 0,
      "center_x" => 540,
      "center_y" => 960
    },
    "enabled" => true,
    "contentDescription" => nil
  },
  [ 1] {
    "class" => "android.widget.LinearLayout",
    "tag" => nil,
    "description" => "android.widget.LinearLayout{e2b638c V.E..... 0,0-1080,1776}",
    "id" => nil,
```

A lista de componentes apresenta um objeto json escrito na tela, ela exibe os atributos (classe, ID, textos, etc.). Para localizar o Lado 1 Executar: `query("* id:'txtLado1'")`

Sintaxe para localizar um componente por ID - `query("* id:'id do componente'")`



```
irb(main):005:0> query("* id:'txtLado1'")
[
  [0] {
    "id" => "txtLado1",
    "enabled" => true,
    "contentDescription" => nil,
    "text" => "",
    "visible" => true,
    "tag" => nil,
    "description" => "android.support.v7.widget.AppCompatEditText{b086d248 VFED..CL .F....
.. 0,29-432,97 #7f0c0051 app:id/txtLado1}",
    "class" => "android.support.v7.widget.AppCompatEditText",
    "rect" => {
      "center_y" => 209,
      "center_x" => 240,
      "height" => 68,
      "y" => 175,
      "width" => 432,
      "x" => 24
    }
  }
]
```

5 – Preenchendo um campo de texto

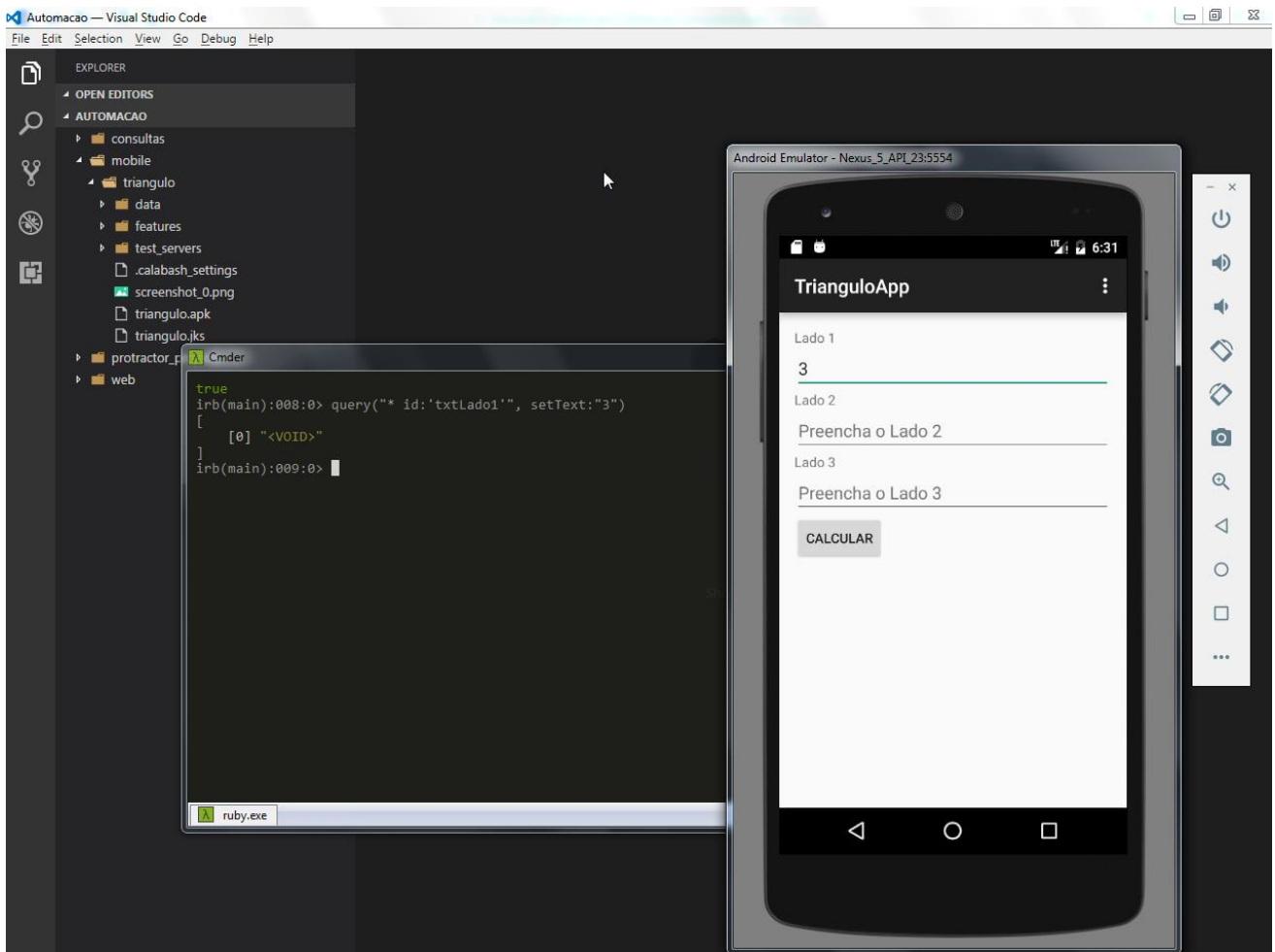
Ao localizar um elemento, podemos interagir com ele, para ver o retorno executar:

```
query("* id:'txtLado1'", setText:"3")
```

O sistema preencheu o lado 1 com o valor de 3

Para função ficar mais bonita podemos executar o comando:

```
enter_text "* id:'txtLado2'", "5"
```

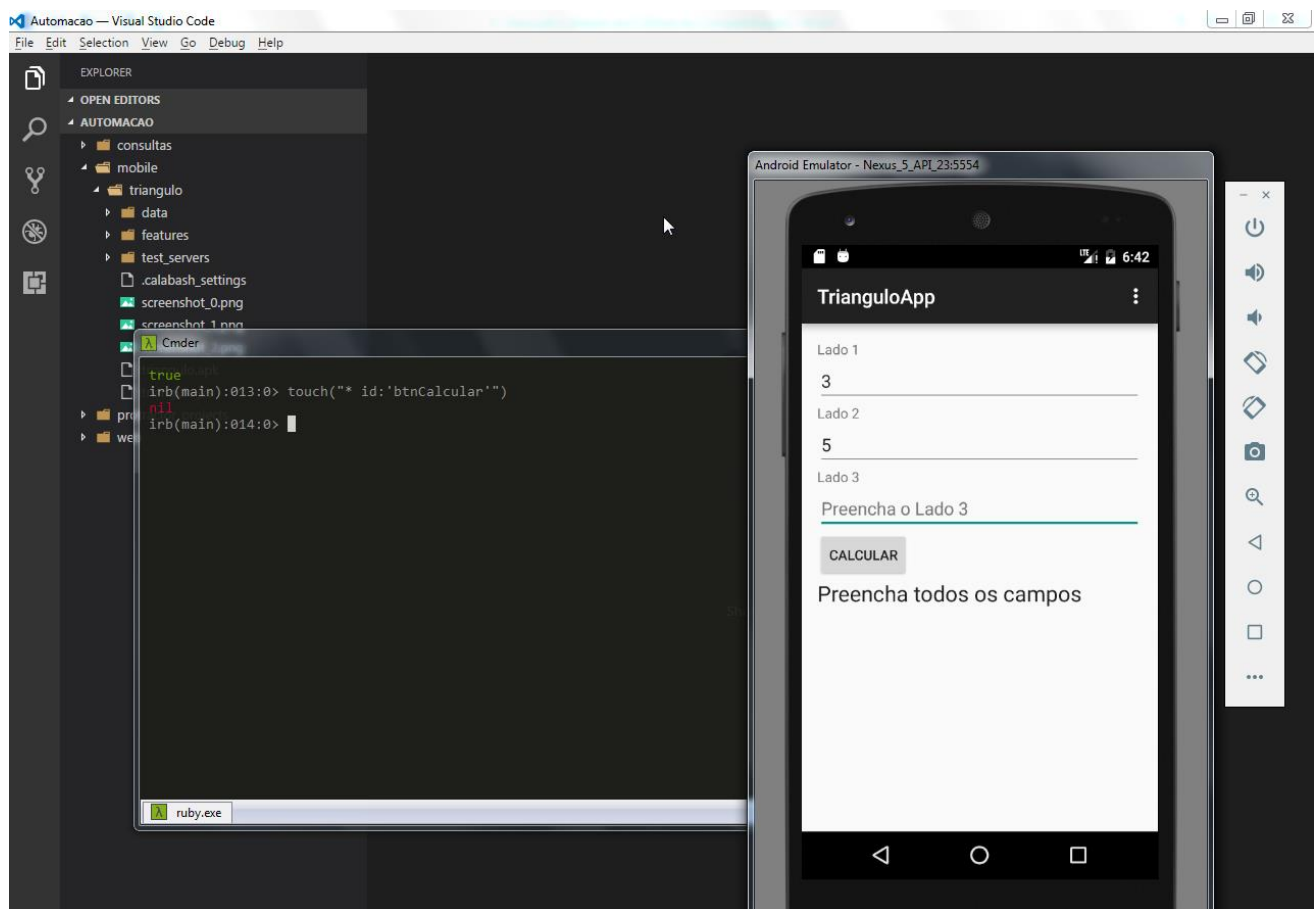


6 – Clicando em componentes

Para clicar usamos o comando touch, também usaremos a localização por ID

Executar: `touch("** id:'btnCalcular'")`

O Sistema clicou e exibiu a mensagem para 'preencher todos os campos'

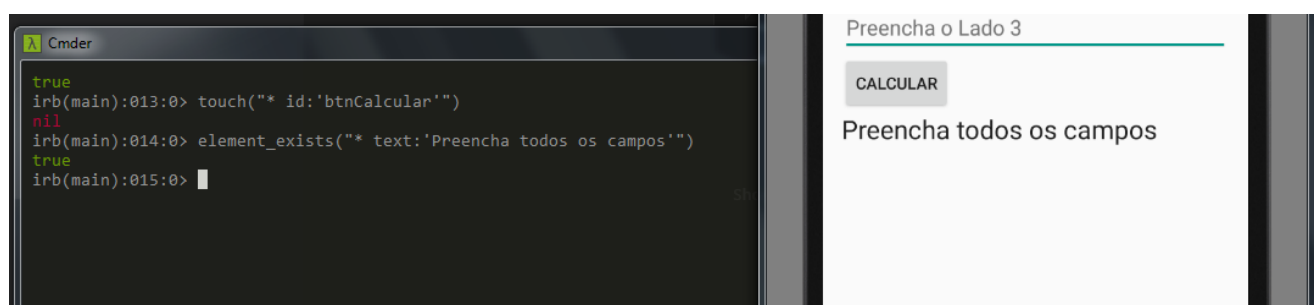


7 – Validando textos

Para validar um resultado exibido na tela por um componente que possui texto, executar:

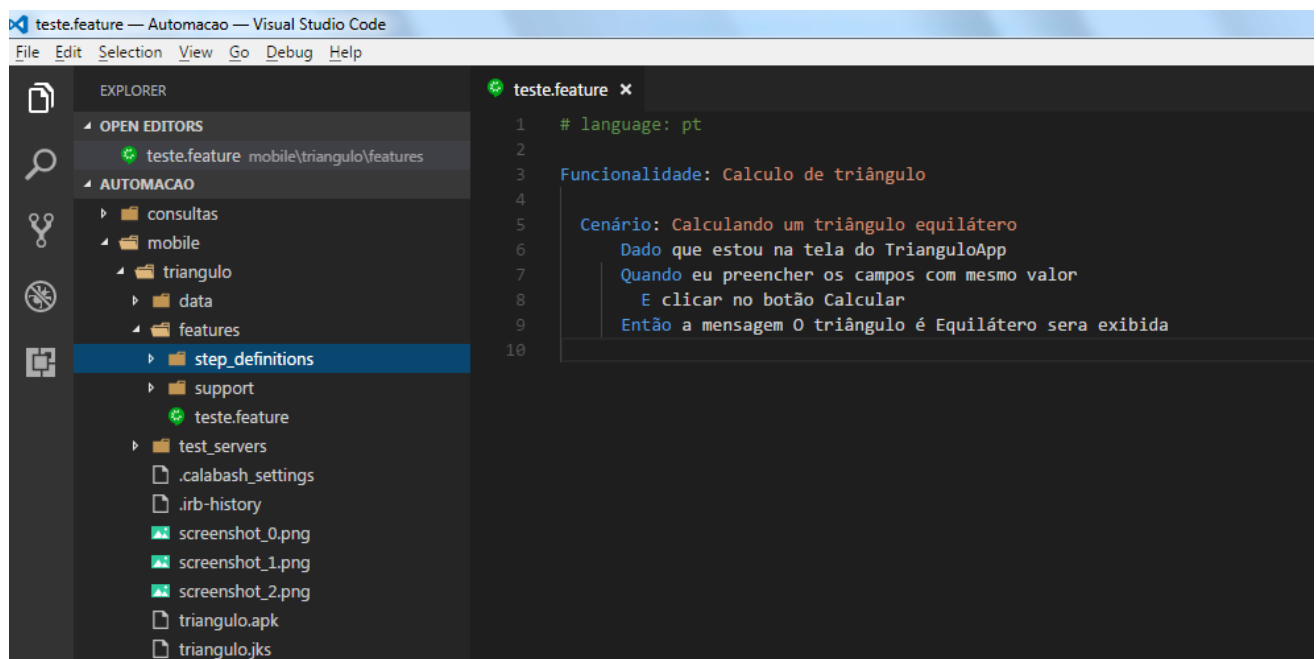
`element_exists("** text:'Preencha todos os campos'")`

Retornou true, verdadeiro, o texto existe na tela



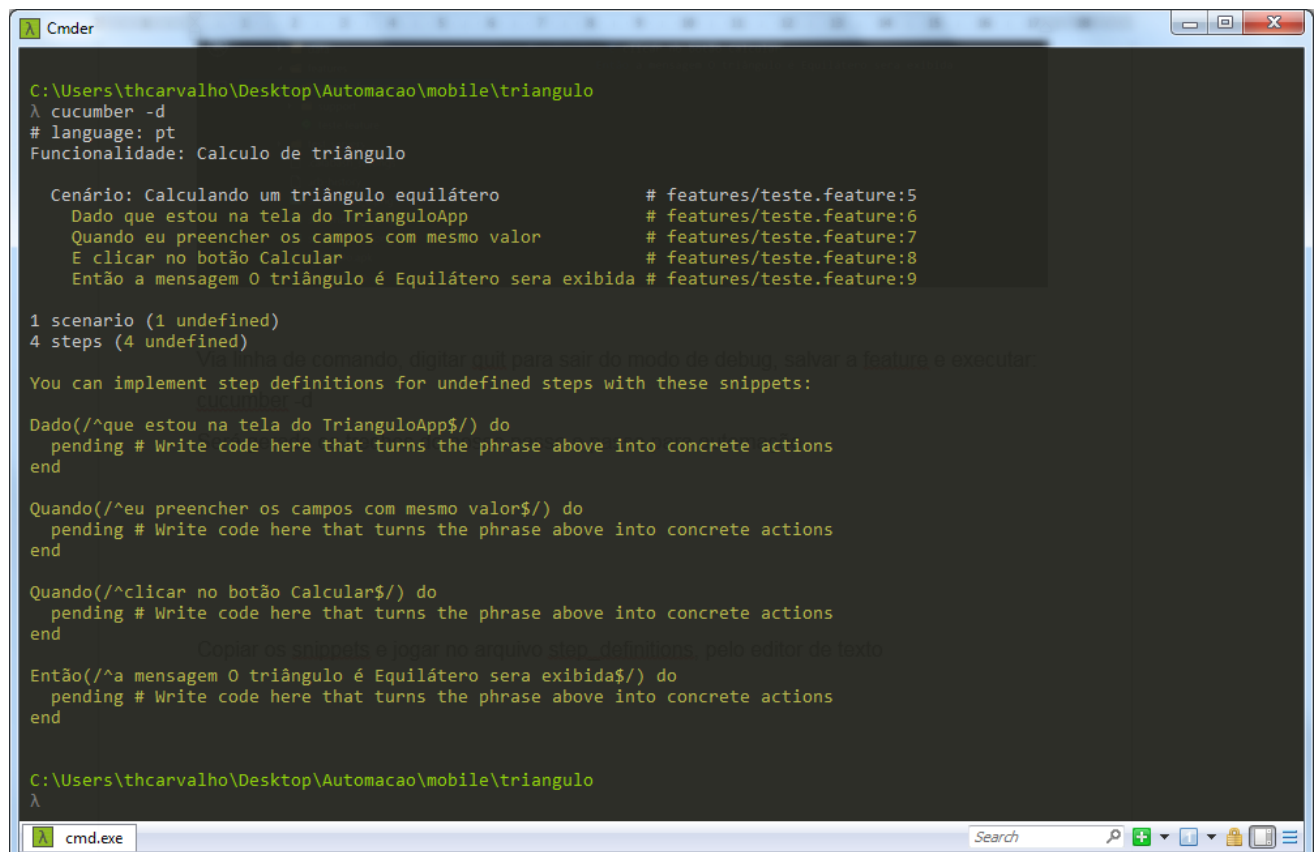
8 – Vamos aos testes?

Abrir o editor de texto, escrever um caso de teste que faça validação do nosso cenário

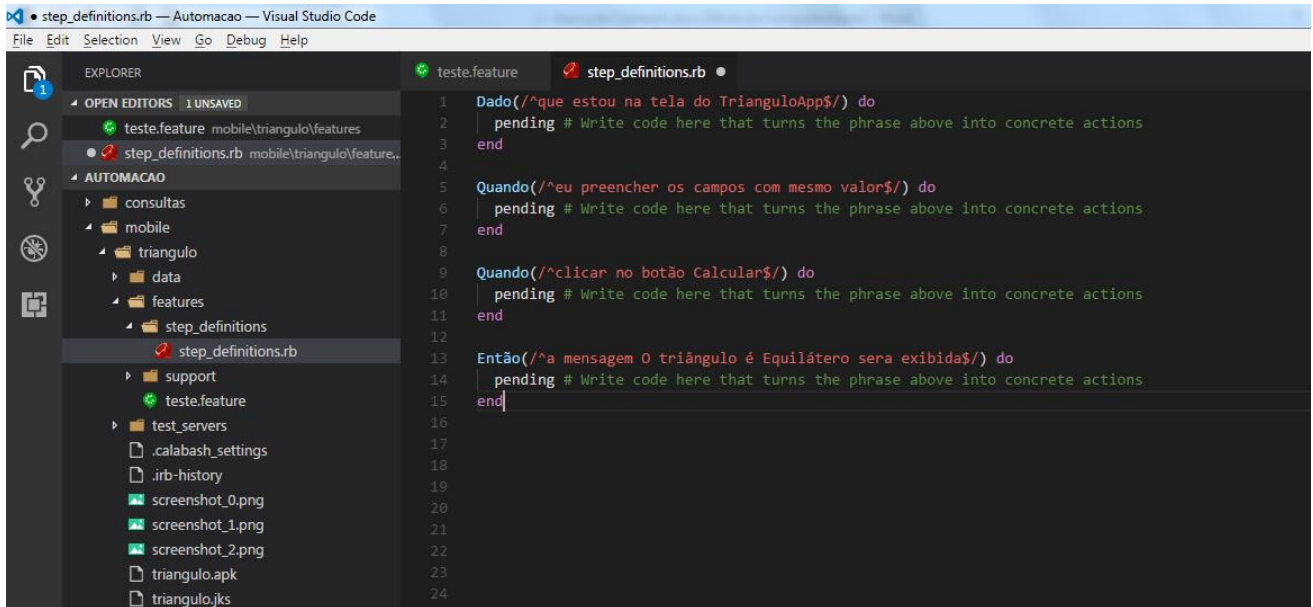


Via linha de comando, digitar quit para sair do modo de debug, salvar a feature e executar:
cucumber -d

Será gerado os trechos do nosso passo a passo para automação



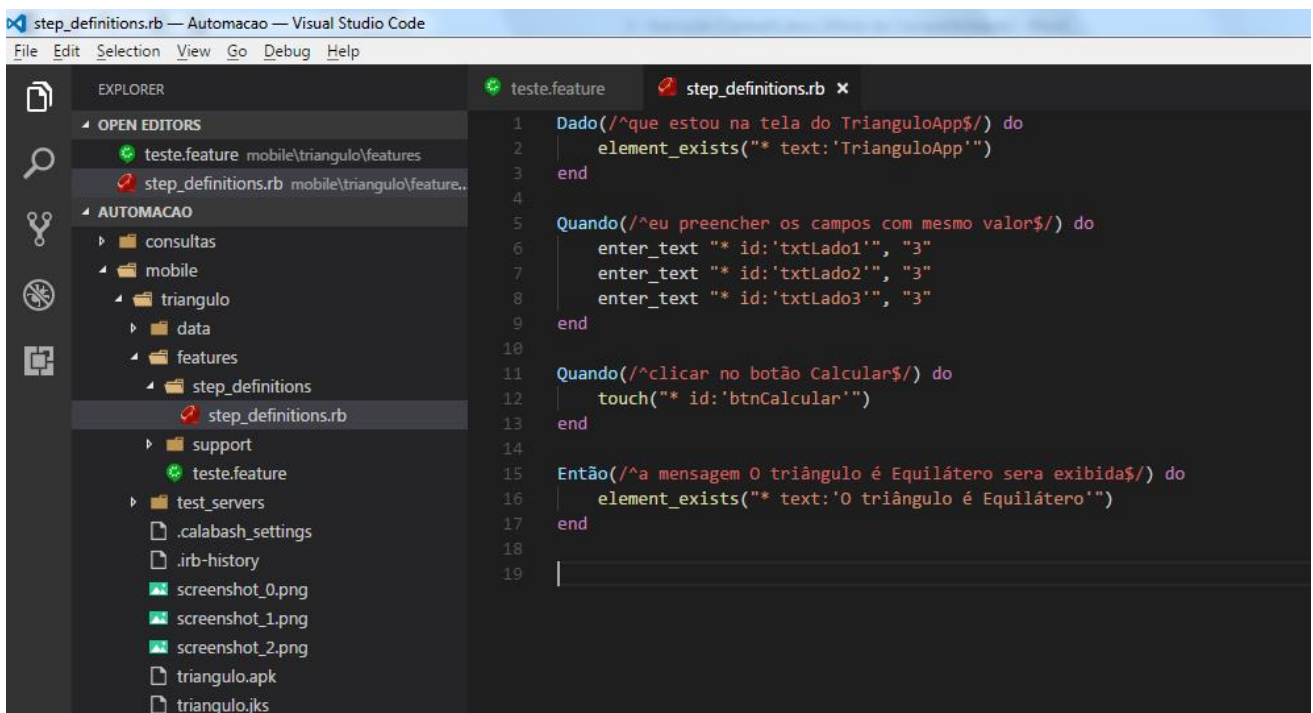
Copiar os snippets e jogar no arquivo step_definitions, pelo editor de texto



The screenshot shows the Visual Studio Code interface with the file explorer on the left and the editor on the right. The file explorer shows the project structure with folders like 'consultas', 'mobile', 'triangulo', 'data', 'features', 'support', and 'test_servers'. The 'step_definitions.rb' file is selected in the 'features' folder. The editor shows the following code:

```
1 Dado(/^que estou na tela do TrianguloApp$/) do
2   pending # Write code here that turns the phrase above into concrete actions
3 end
4
5 Quando(/^eu preencher os campos com mesmo valor$/) do
6   pending # Write code here that turns the phrase above into concrete actions
7 end
8
9 Quando(/^clicar no botão Calcular$/) do
10  pending # Write code here that turns the phrase above into concrete actions
11 end
12
13 Então(/^a mensagem O triângulo é Equilátero sera exibida$/) do
14  pending # Write code here that turns the phrase above into concrete actions
15 end
```

Vamos implementar nossos testes, usando os comandos que vimos até agora



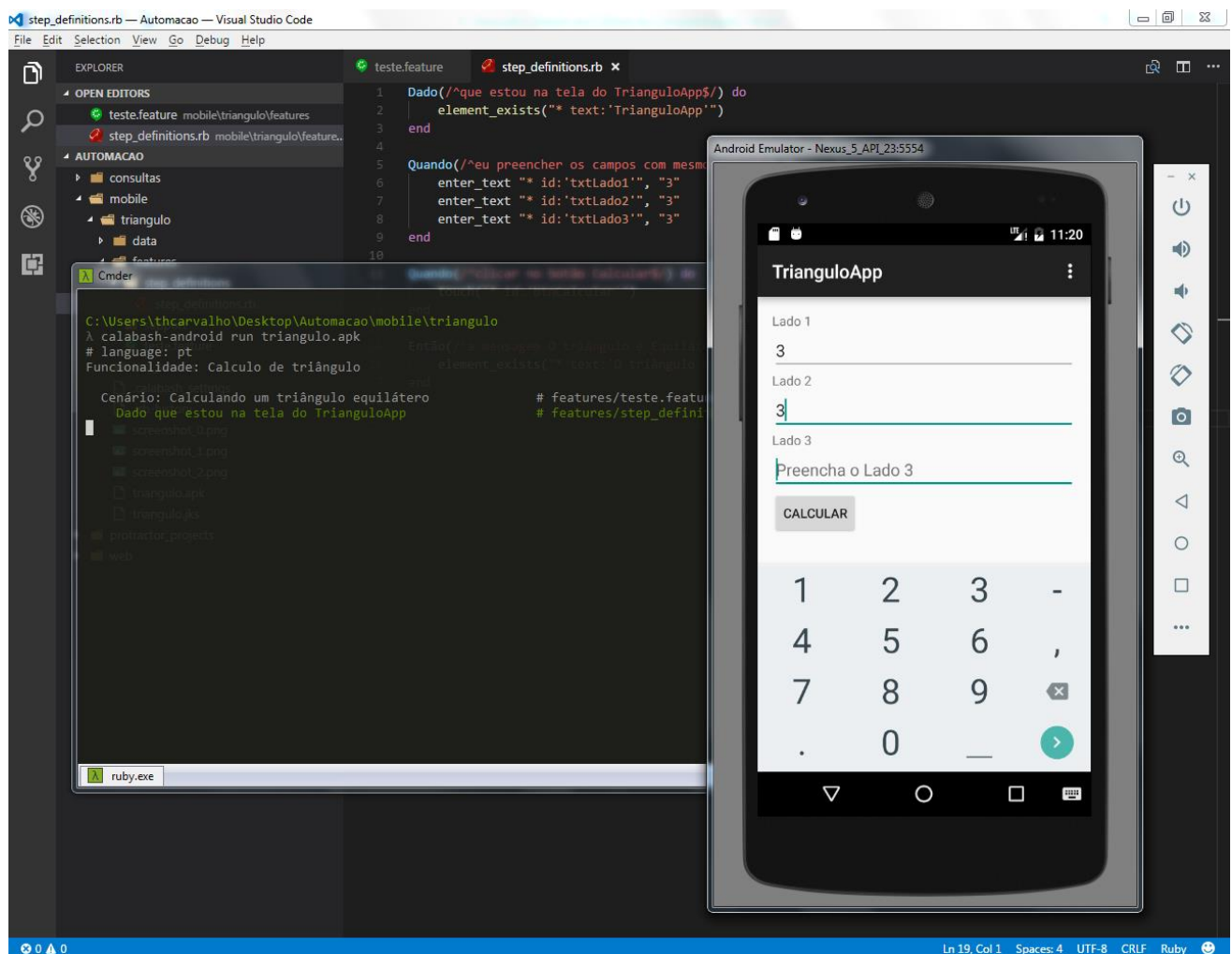
The screenshot shows the Visual Studio Code interface with the file explorer on the left and the editor on the right. The file explorer shows the project structure with folders like 'consultas', 'mobile', 'triangulo', 'data', 'features', 'support', and 'test_servers'. The 'step_definitions.rb' file is selected in the 'features' folder. The editor shows the following code:

```
1 Dado(/^que estou na tela do TrianguloApp$/) do
2   element_exists("* text:'TrianguloApp'")
3 end
4
5 Quando(/^eu preencher os campos com mesmo valor$/) do
6   enter_text "* id:'txtLado1'", "3"
7   enter_text "* id:'txtLado2'", "3"
8   enter_text "* id:'txtLado3'", "3"
9 end
10
11 Quando(/^clicar no botão Calcular$/) do
12   touch("* id:'btnCalcular'")
13 end
14
15 Então(/^a mensagem O triângulo é Equilátero sera exibida$/) do
16   element_exists("* text:'O triângulo é Equilátero'")
17 end
18
19
```

9 – Executando os cenários

Salvar nosso caso de teste, e via console executar: calabash-android run nomedoapp.apk

O sistema irá realizar automaticamente o cálculo seguindo os comandos da step



Resultado da execução dos testes

