Kranberry

Nossa biblioteca desenvolvida para facilitar a automação de testes de UI nativos Android

Sobre nós

Há mais de 13 anos no mercado de TI e atualmente trabalhando como QA, possui experiência com desenvolvimento Backend e como Tech Lead. Atualmente focada em aplicativos Mobile Nativos.

Ana Ludmila

@ThoughtWorks

Trabalha com desenvolvimento Mobile há 9 anos. Hoje em dia, faz parte do programa Google Developer Experts, na tecnologia Android.

Angélica Oliveira

@Spotify

Kranberry

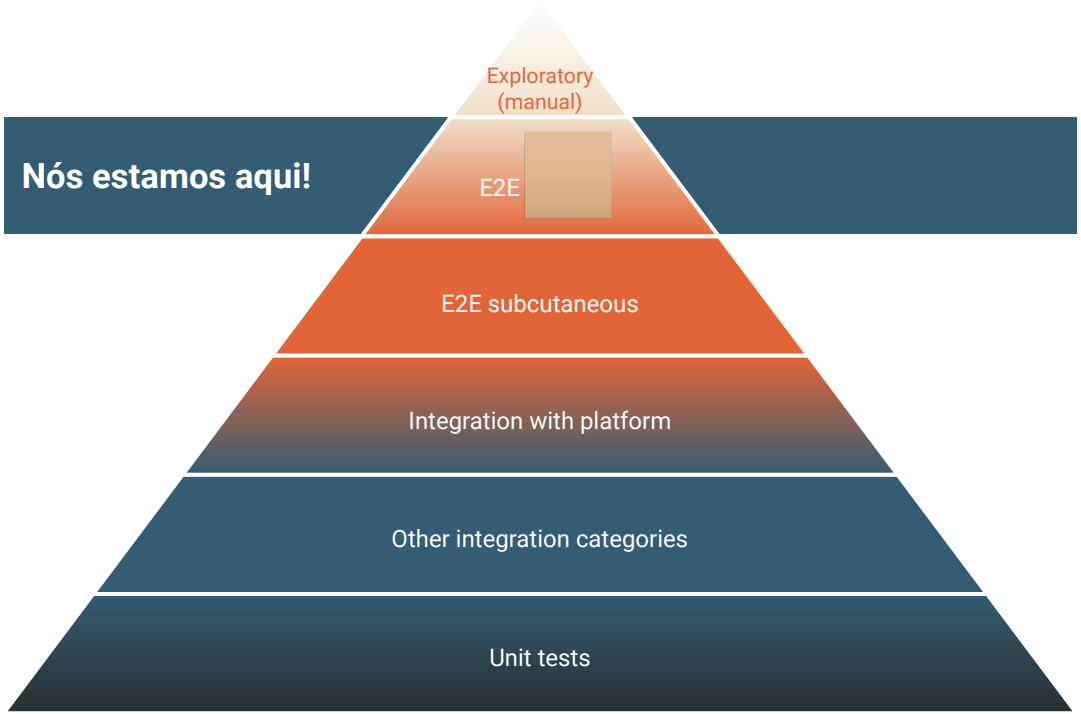
Agenda

Contexto/Problema

Automatizar a maior parte da Pirâmide de Testes Ideal

Valor de negócio

Custos de manutenção



Por que criar uma biblioteca?

Em um contexto onde a melhor solução ponderada é implementar testes na stack Android Nativa, existem uma série de tarefas e problemas que precisamos lidar.

Compreendendo que em produtos similares as tarefas são repetitivas, idealizamos a biblioteca extraindo essas necessidades em comum.

Optamos pela stack Android Nativa

Demoramos pelo menos 1 sprint para configurar o pacote de testes, utilizando o UIAutomator

Tivemos que aprender a lidar com algumas particularidades do framework

Precisamos repetir as configurações em outro produto nativo

Kranberry

Quais atividades foram consideradas repetitivas?

Lidar com permissões do Device

Entender e implementar uma dinâmica que permita ao pacote de testes manipular os arquivos necessários.

Definir dependências / Arquitetura

Apesar de conhecer o Robot Framework, optamos por definir uma arquitetura similar. Além disso, precisamos identificar as compatibilidades entre dependências utilizadas.

Lidar com geração de evidências de teste (screenshots)

Entendemos que em alguns contextos, evidências visuais da execução dos testes são exigidas por questões de compliance. O que também era o nosso caso.

Automação das tasks de integração com as pipelines

Como toda automação, o objetivo principal desta camada de testes é execução sem intervenções manuais, o que exigem tarefas automatizadas que possam ser acionadas através de triggers em pipelines.

Kranberry

Quais atividades foram consideradas repetitivas?

Lidar com feedback dos testes na pipeline (command line)

Ter um feedback claro e efetivo sobre os passos de execução dos testes, bem como possíveis erros para direcionar o debug.

Lidar com testes guiados por dados externos

Obtenção de massa de dados via arquivos externos, inicialmente utilizando jsons (para melhor integração com apis rest/mocks) e estendendo para outras possíveis formas (csv, txt, etc.)

Gerenciar parametrizações de forma centralizada (timeouts, appPackages, etc)

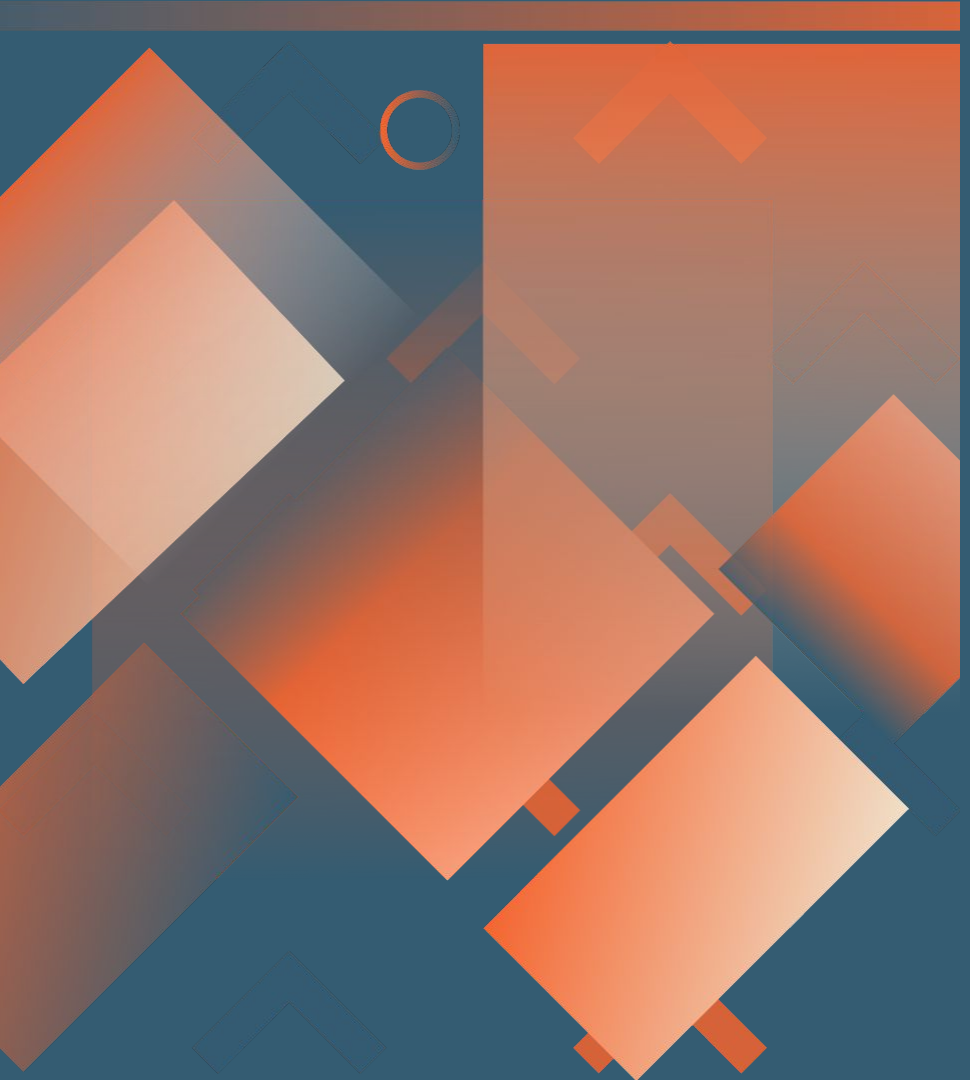
A partir do momento em que precisamos replicar configurações para outros produtos, fazer-lo via intervenções no código é pouco produtivo. A melhor alternativa seria obter estes parâmetros através de um arquivo.

Problemas de performance com alguns recursos do UiAutomator (scroll por exemplo)

Lidar com esperas explícitas e implícitas de maneira adequada. Também precisamos ter alternativas sobre alguns recursos do UIAutomator os quais consideramos de baixa performance.

Então criamos o Kranberry!

Kranberry

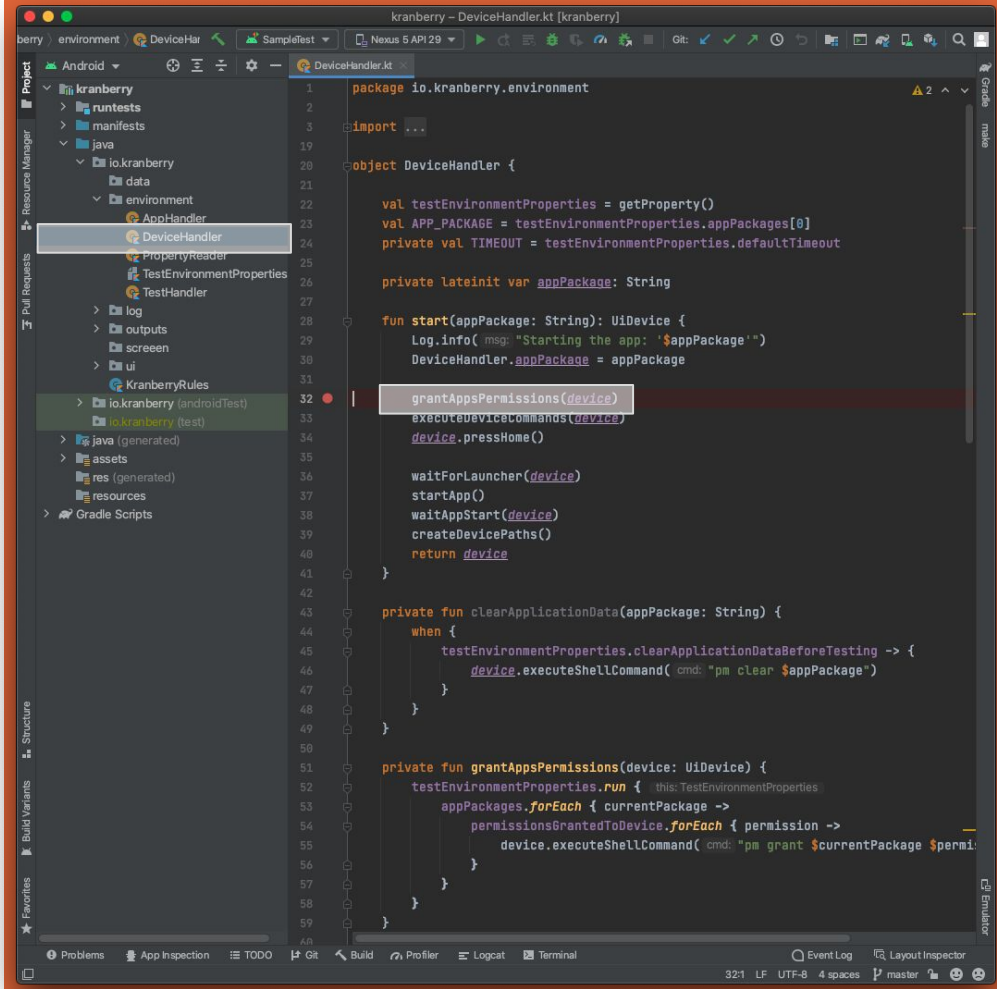


Cada tarefa repetitiva foi transformada em uma funcionalidade na Biblioteca!

Lidar com permissões do Device

io.kranberry.environment.DeviceHandler

Alguns aplicativos necessitam de permissões do device para realizar determinadas funcionalidades. Tratamos essa questão configurando o pedido de permissões de forma automatizada, quando os testes são executados.

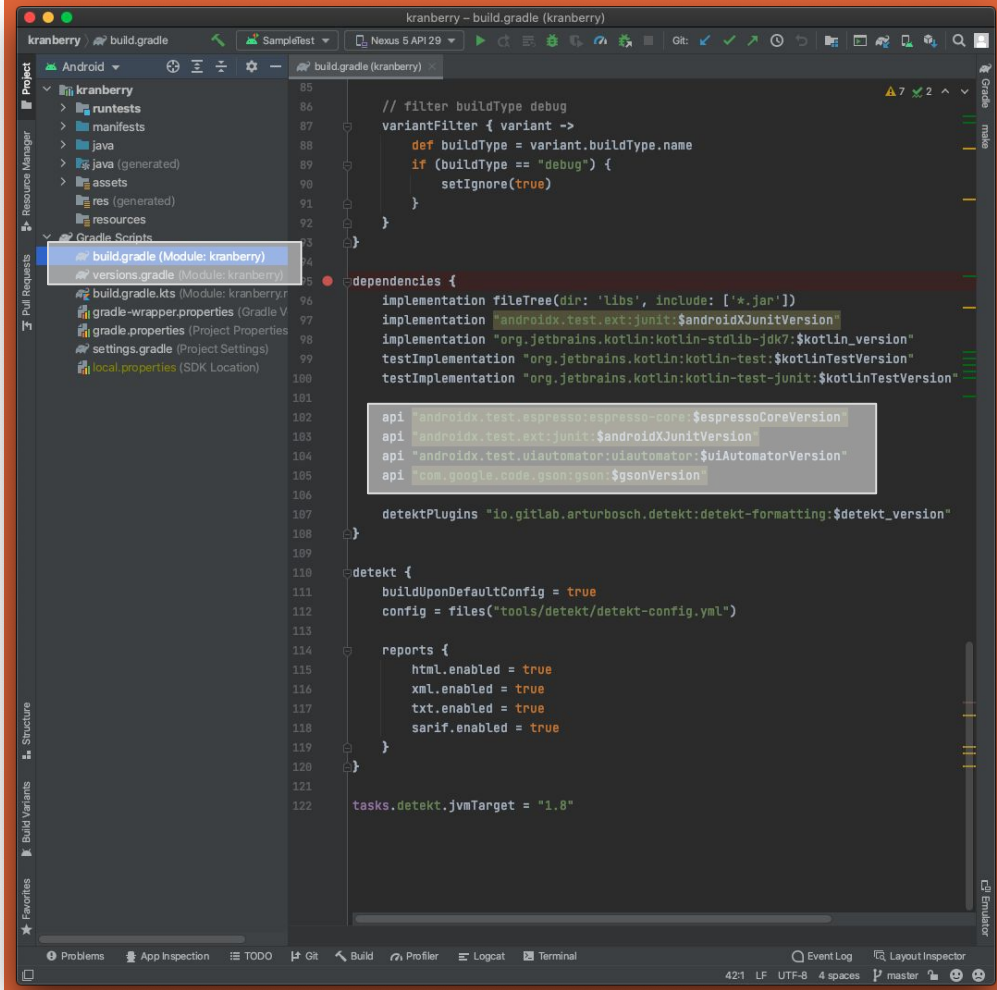


Kranberry

Definir dependências / Arquitetura

kranberry/build.gradle

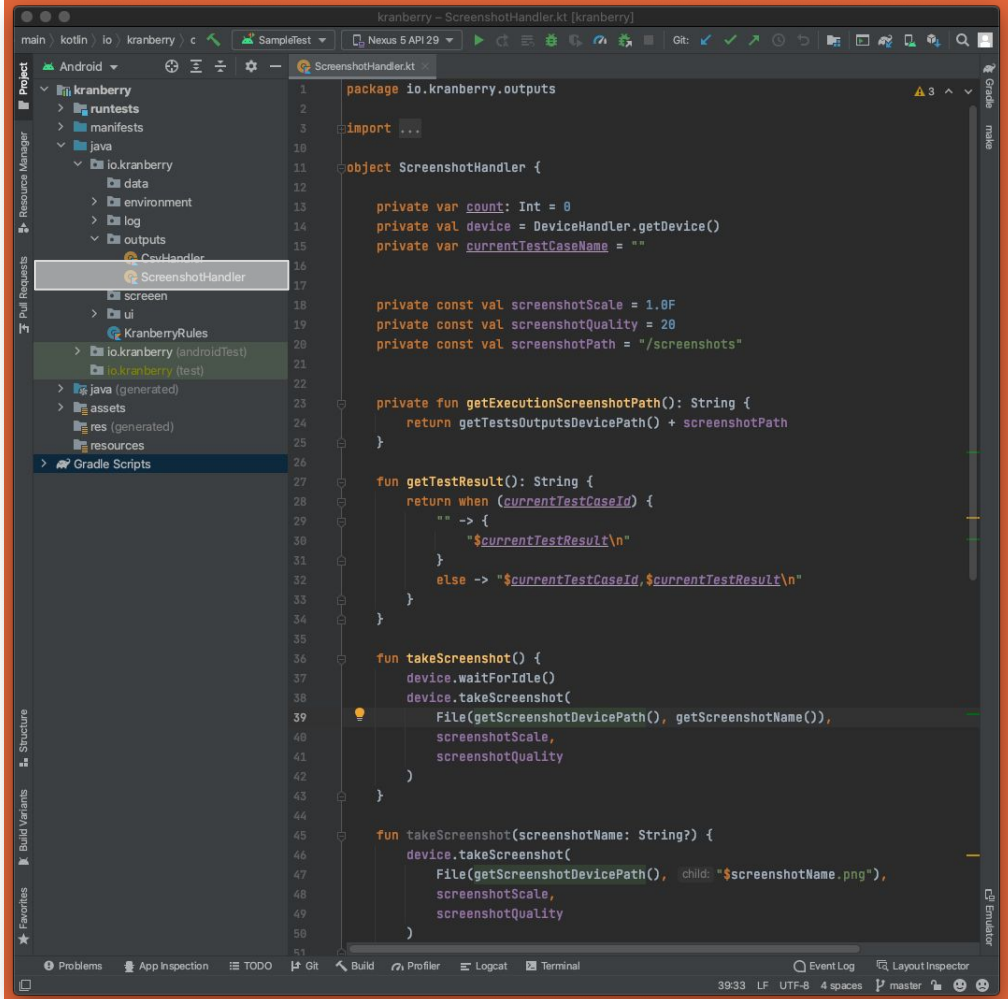
Utilizamos algumas dependências para executar e configurar nossos testes, como UIAutomator, JUnit, Espresso e Google GSON.



Lidar com geração de evidências de teste (screenshots)

io.kranberry.environment.ScreenshotHandler

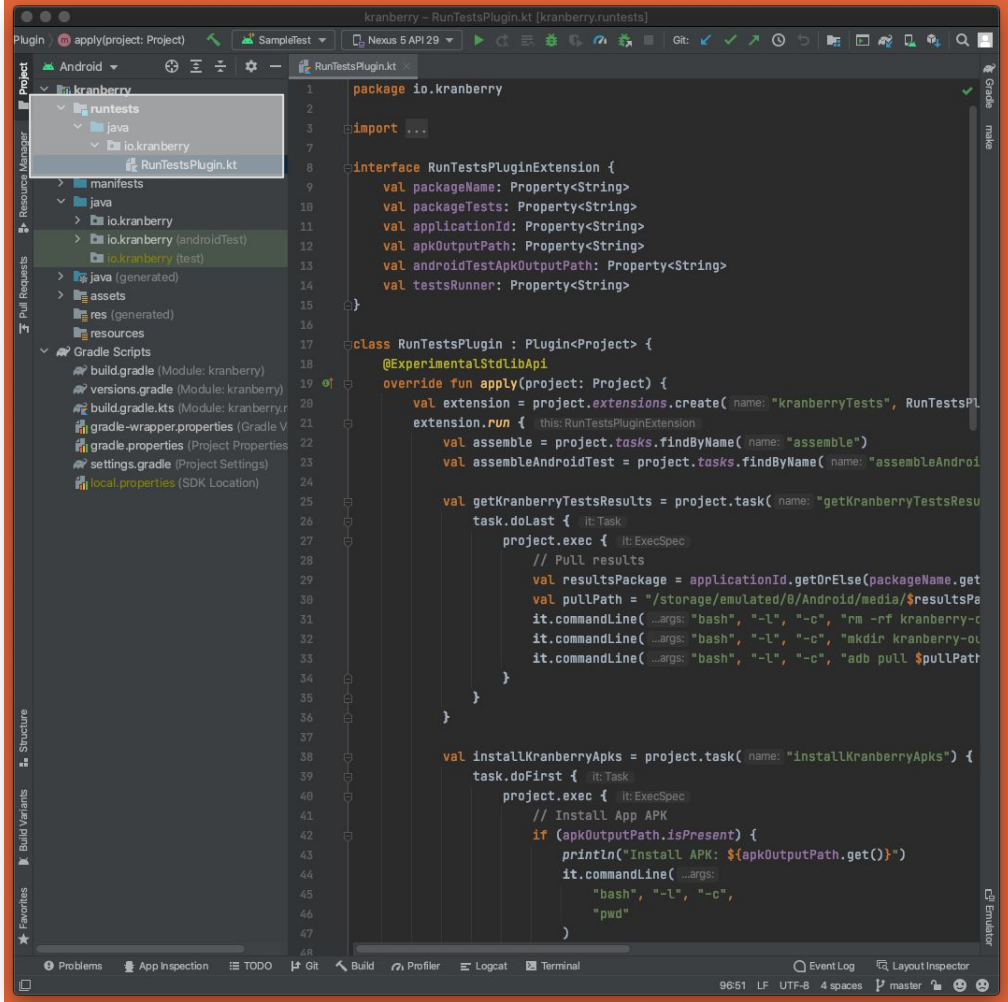
Geração de screenshots como evidências de sucesso ou falha, sendo armazenados com nomes específicos ou no caso de não fornecermos um nome, é utilizado o nome do teste sob execução.



Automação das tasks de integração com as pipelines

[ ] RunTestsPlugin

Disponibilizamos um Plugin do Gradle, que caso seja adicionado e configurado no módulo em que os testes com a lib Kranberry estejam implementados, pode facilitar bastante a execução dos mesmos pelo terminal. Facilitando assim, a execução de testes em um ambiente de CI.

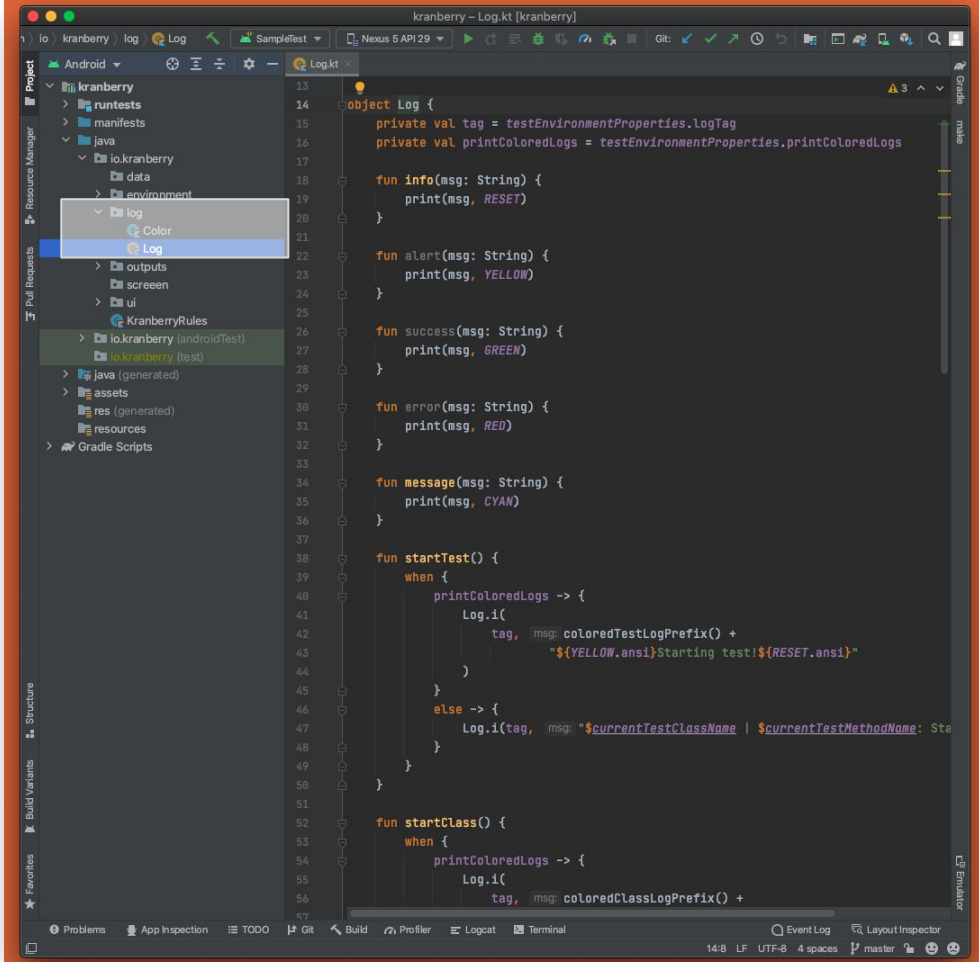


Kranberry

Lidar com feedback dos testes na pipeline (command line)

package io.kranberry.log

Foi implementada uma estrutura de log, que fornece feedback em tempo real na linha de comando, podendo ser coloridos ou não dependendo do parâmetro informado no properties.



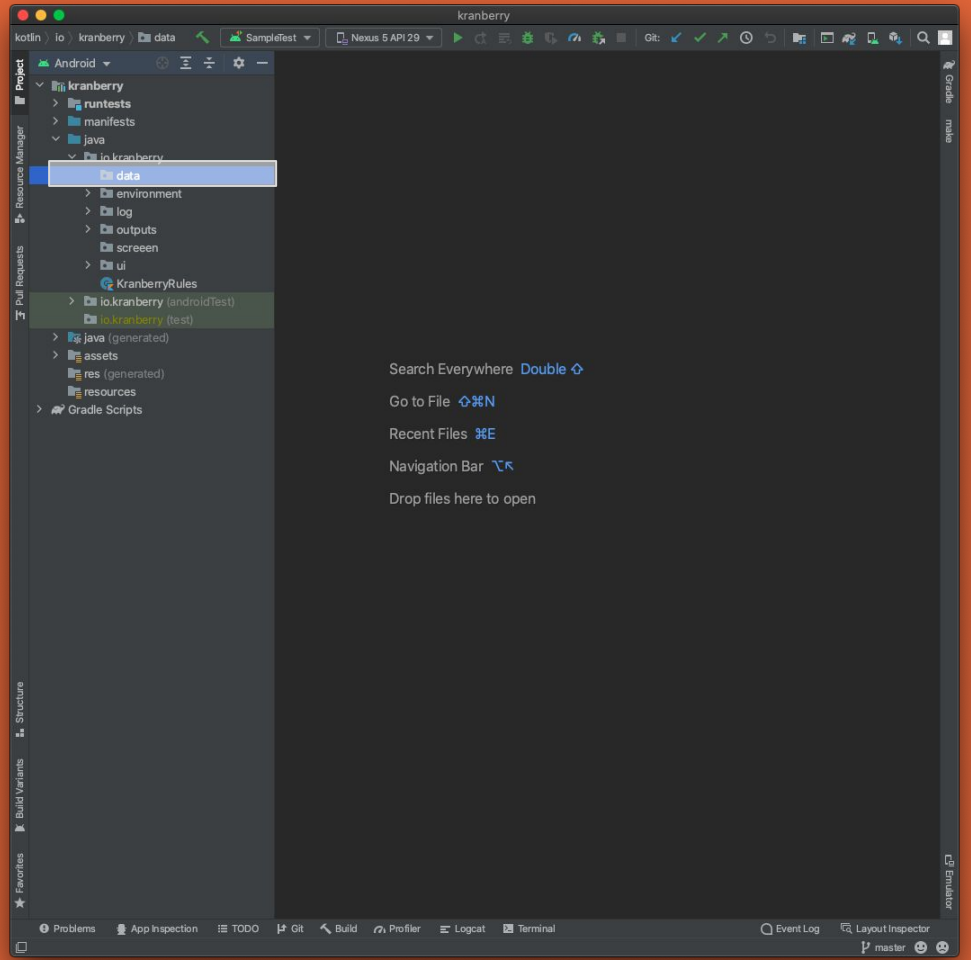
Lidar com testes guiados por dados externos

package io.kranberry.data

⚠️ 👷🏼 👷🏾 🛠️

Está no Roadmap!

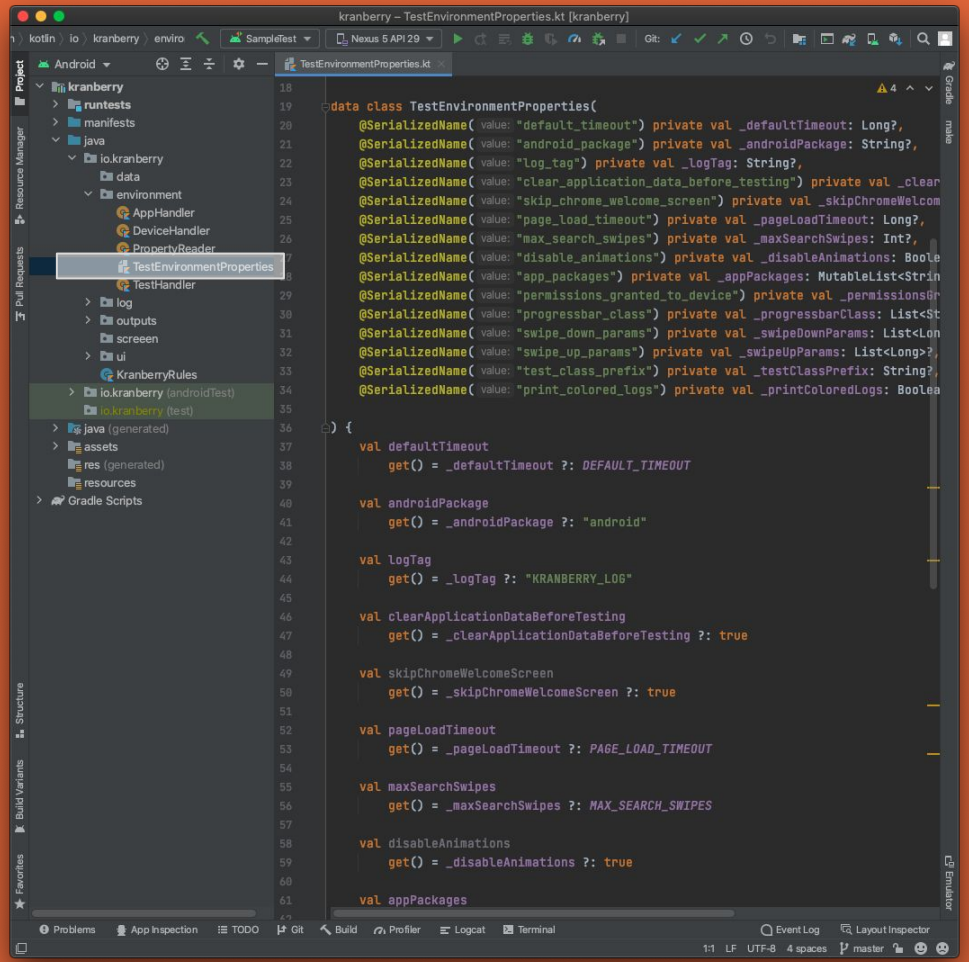
Ainda sim, já temos uma estrutura de leitura de arquivos externos, precisamos agora concluir a implementação!



Gerenciar parametrizações de forma centralizada

io.kranberry.environment.TestEnvironmentProperties

Uma maneira de ler dinamicamente os parâmetros que diferem de maneira genérica os pacotes de teste.

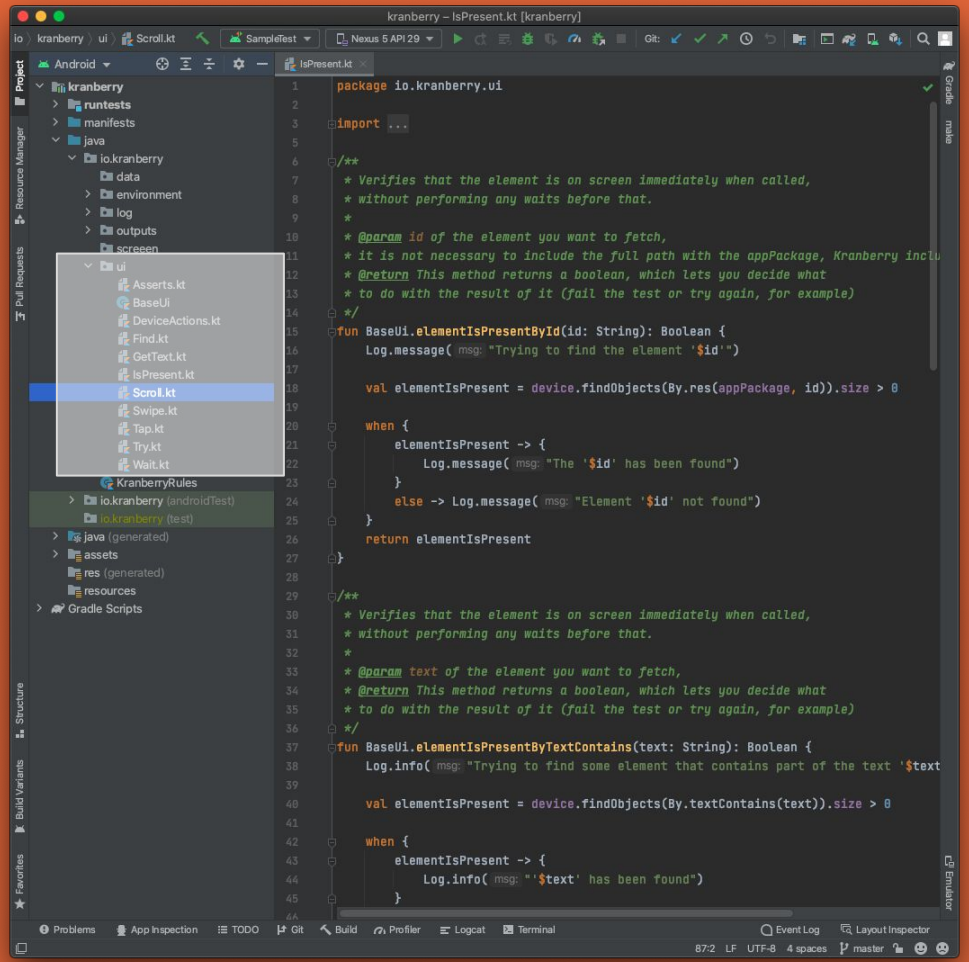


Kranberry

Problemas de performance com alguns recursos do UiAutomator

io.kranberry.ui

Otimização de esperas explícitas e implícitas, de manipulação de elementos da UI e documentação baseada em melhores práticas de utilização para cada caso.



Ainda não terminou...

Além das funcionalidades apresentadas até aqui, também envelopamos vários comandos de manipulação de elementos da UI, com o intuito de gerar logs mais legíveis e tornar a implementação dos testes mais fáceis!

```
tapById(id: String): Boolean
```

```
tapByClass(className: String): Boolean
```

```
elementIsPresentById(id: String): Boolean
```

```
elementIsPresentByTextContains(text: String): Boolean
```

Getting Started!

Quero começar a usar o Kranberry!

Pré requisitos:

- Testes de UI (Instrumentados)
- Kotlin
- Seguir passo-a-passo da documentação!
- Se tiver dúvidas, consulte os exemplos, ou pode contar com a gente! ;)

Documentação completa no **Github**

Projeto de exemplo: **kranberry-sample**

Com o Kranberrry, você é capaz de configurar seu pacote de testes E2E em minutos!

00:01:32

```
warning: Pulling without specifying how to reconcile divergent branches is
discouraged. You can squelch this message by running one of the following
commands sometime before your next pull:

  git config pull.rebase false  # merge (the default strategy)
  git config pull.rebase true   # rebase
  git config pull.ff only       # fast-forward only

You can replace "git config" with "git config --global" to set a default
preference for all repositories. You can also pass --rebase, --no-rebase,
or --ff-only on the command line to override the configured default per
invocation.

Already up to date.

~/Documents/kranberry-io/kranberry-sample on ᚠ master 15:10:22
$ git checkout -b tested-by-kranberry
Switched to a new branch 'tested-by-kranberry'

~/Documents/kranberry-io/kranberry-sample on ᚠ tested-by-kranberry 15:10:31
$ git add .
```

E o futuro?

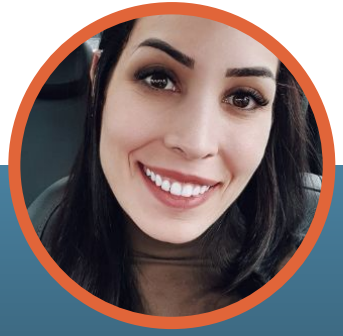
Novas funcionalidades + estabilidade

Roadmap, idéias...

Para o futuro desejamos estabilizar as funcionalidades já existentes além de implementar novas funções na biblioteca com base em feedbacks da comunidade! Também desejamos automatizar mais os processos que hoje são manuais.

Contribua!

Agradecimento especial ❤️



Kelly Barros
@ThoughtWorks

Rafael Benevenuto
@ThoughtWorks

Aline Ayres
@ThoughtWorks

Tamiris Fernandes
@ThoughtWorks

Simão Pedro
@ThoughtWorks

Até breve!
Perguntas?