



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.

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



Ana
Ludmila

@ThoughtWorks

Angélica
Oliveira

@Spotify



Agenda

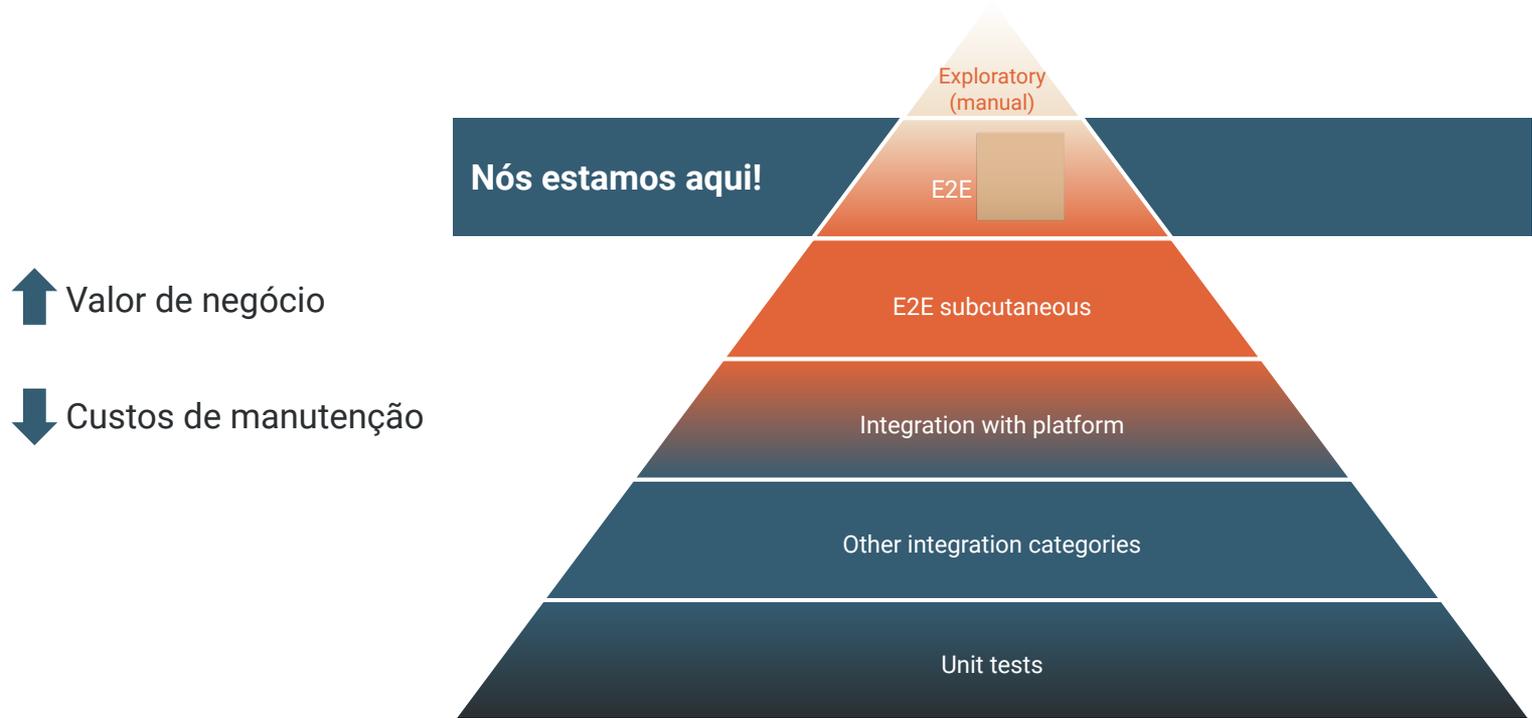
- | | | | |
|----|---|------------|-----------|
| 1. | Contexto/Problema | 5 minutos | <u>05</u> |
| 2. | Criação do Kranberry! | 2 minutos | <u>09</u> |
| 3. | Funcionalidades da biblioteca | 10 minutos | <u>10</u> |
| | ■ Permissões de device/emulador | | <u>11</u> |
| | ■ Dependências & Arquitetura | | <u>12</u> |
| | ■ Screenshots | | <u>13</u> |
| | ■ Plugin Gradle - Automação de tarefas | | <u>14</u> |
| | ■ Feedback - Terminal | | <u>15</u> |
| | ■ Testes Guiados Por Dados | | <u>16</u> |
| | ■ Parametrização | | <u>17</u> |
| | ■ Performance & Outros | | <u>18</u> |
| 4. | Getting Started! | 2 minutos | <u>20</u> |
| | ■ Pré-Requisitos & Documentação | | <u>21</u> |
| 5. | Vídeo - Testes E2E em 15 minutos | 6 minutos | <u>22</u> |
| 6. | E o futuro? | 2 minutos | <u>24</u> |
| 7. | Como contribuir! | 2 minutos | <u>25</u> |
| 8. | Perguntas | 6 minutos | <u>26</u> |



Contexto/Problema



Automatizar a maior parte da Pirâmide de Testes Ideal



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 extraíndo 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

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.

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)

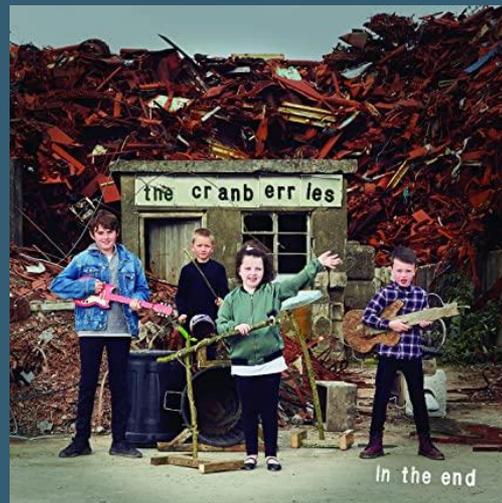
A partir do momento em que precisamos replicar configurações para outros produtos, fazer-lo via intervenções no código é pouco produtivo. A melhor alternativa seria obter estes parâmetros através de um arquivo.

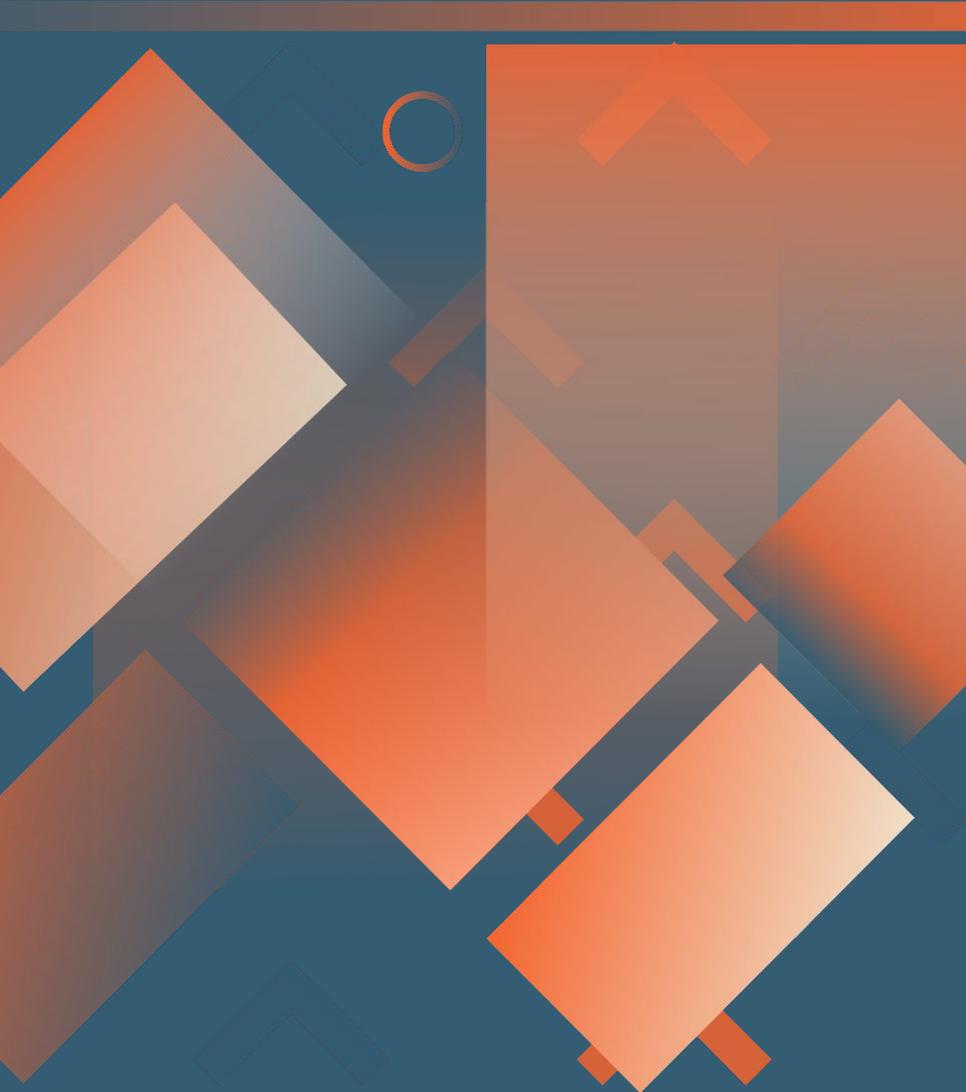
Problemas de performance com alguns recursos do UiAutomator (scroll por exemplo)

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



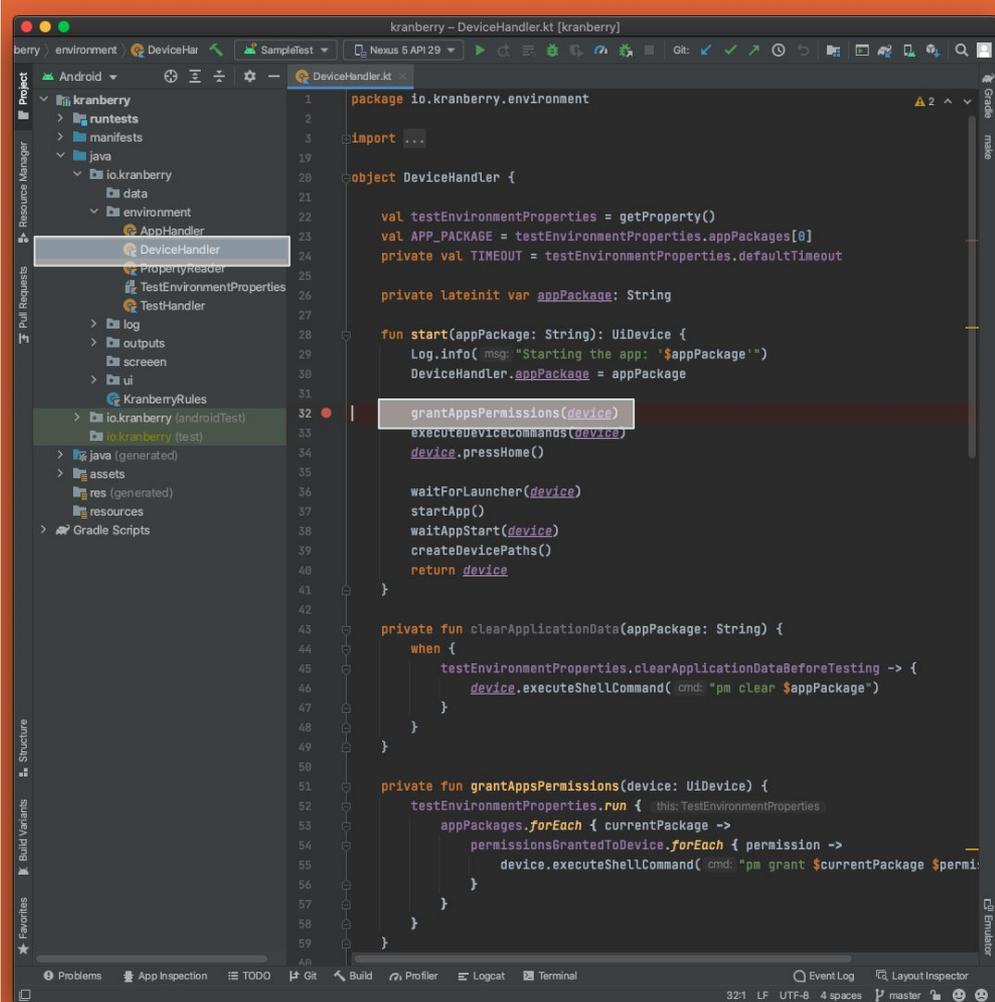
The background features a dark blue field with several overlapping, semi-transparent geometric shapes in various shades of orange and light brown. These shapes include triangles, squares, and rectangles, some of which are oriented at angles. A small, thin orange circle is positioned in the upper left quadrant. The overall aesthetic is modern and graphic.

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.



```
package io.kranberry.environment

import ...

object DeviceHandler {

    val testEnvironmentProperties = getProperty()
    val APP_PACKAGE = testEnvironmentProperties.appPackages[0]
    private val TIMEOUT = testEnvironmentProperties.defaultTimeout

    private lateinit var appPackage: String

    fun start(appPackage: String): UiDevice {
        Log.info(msg: "Starting the app: $appPackage")
        DeviceHandler.appPackage = appPackage

        grantAppsPermissions(device)
        executeDeviceCommands(device)
        device.pressHome()

        waitForLauncher(device)
        startApp()
        waitAppStart(device)
        createDevicePaths()
        return device
    }

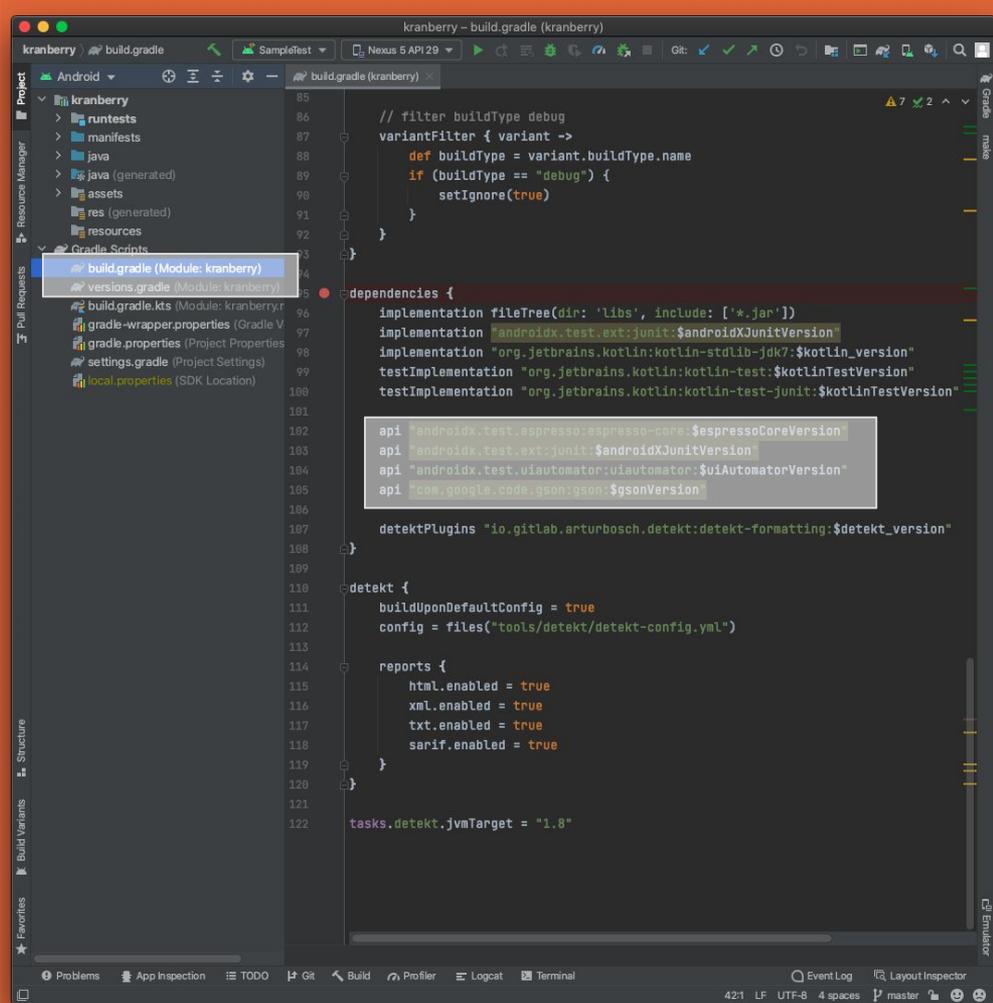
    private fun clearApplicationData(appPackage: String) {
        when {
            testEnvironmentProperties.clearApplicationDataBeforeTesting -> {
                device.executeShellCommand(cmd: "pm clear $appPackage")
            }
        }
    }

    private fun grantAppsPermissions(device: UiDevice) {
        testEnvironmentProperties.run { this: TestEnvironmentProperties
            appPackages.forEach { currentPackage ->
                permissionsGrantedToDevice.forEach { permission ->
                    device.executeShellCommand(cmd: "pm grant $currentPackage $permi
                }
            }
        }
    }
}
```

Definir dependências / Arquitetura

kranberry/build.gradle

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



```
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122

// filter buildType debug
variantFilter { variant ->
    def buildType = variant.buildType.name
    if (buildType == "debug") {
        setIgnore(true)
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation "androidx.test.ext:junit:$androidXJUnitVersion"
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"
    testImplementation "org.jetbrains.kotlin:kotlin-test:$kotlinTestVersion"
    testImplementation "org.jetbrains.kotlin:kotlin-test-junit:$kotlinTestVersion"

    api "androidx.test.espresso:espresso-core:$espressoCoreVersion"
    api "androidx.test.ext:junit:$androidXJUnitVersion"
    api "androidx.test.uiautomator:uiautomator:$uiAutomatorVersion"
    api "com.google.code.gson:gson:$gsonVersion"

    detektPlugins "io.gitlab.arturbosch.detekt:detekt-formatting:$detekt_version"
}

detekt {
    buildUponDefaultConfig = true
    config = files("tools/detekt/detekt-config.yml")

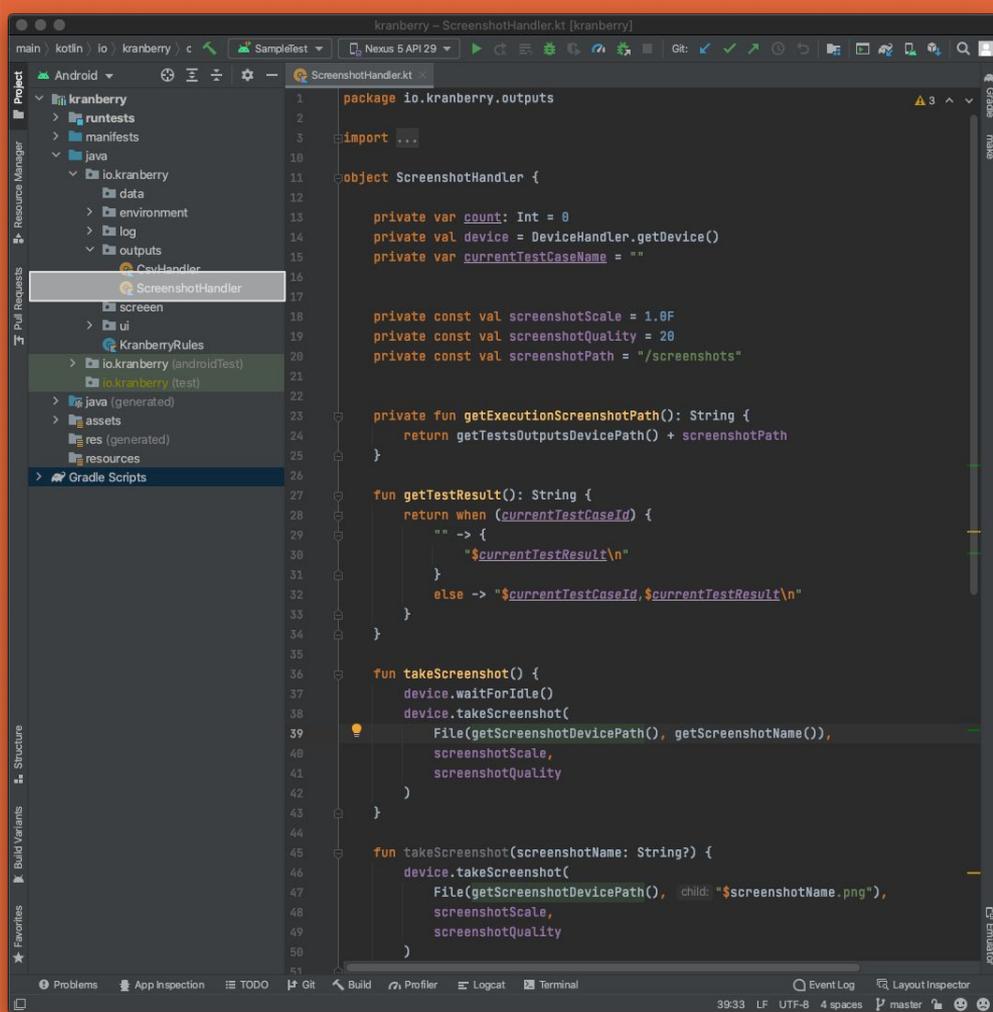
    reports {
        html.enabled = true
        xml.enabled = true
        txt.enabled = true
        sarif.enabled = true
    }
}

tasks.detekt.jvmTarget = "1.8"
```

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.



```
package io.kranberry.outputs

import ...

object ScreenshotHandler {

    private var count: Int = 0
    private val device = DeviceHandler.getDevice()
    private var currentTestCaseName = ""

    private const val screenshotScale = 1.0F
    private const val screenshotQuality = 20
    private const val screenshotPath = "/screenshots"

    private fun getExecutionScreenshotPath(): String {
        return getTestsOutputsDevicePath() + screenshotPath
    }

    fun getTestResult(): String {
        return when (currentTestCaseId) {
            "" -> {
                "$currentTestResult\n"
            }
            else -> "$currentTestCaseId,$currentTestResult\n"
        }
    }

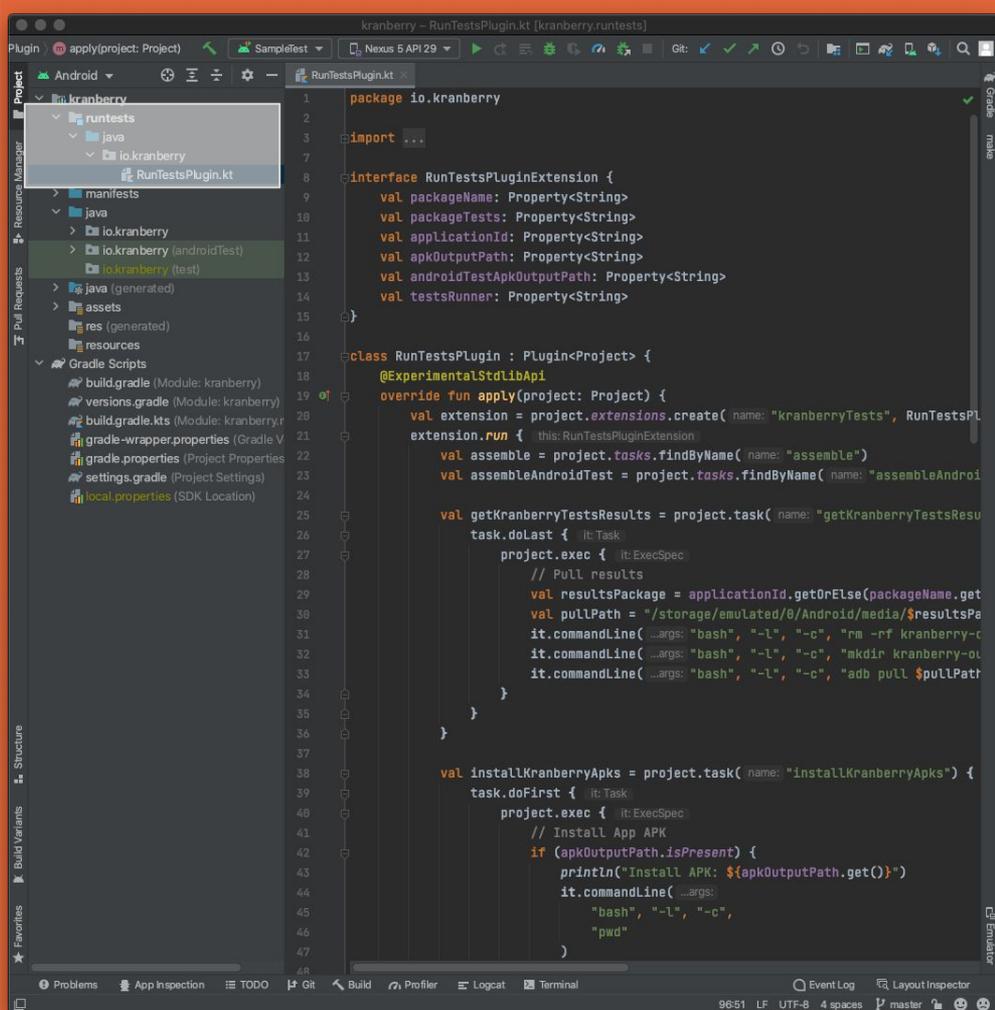
    fun takeScreenshot() {
        device.waitForIdle()
        device.takeScreenshot(
            File(getScreenshotDevicePath(), getScreenshotName()),
            screenshotScale,
            screenshotQuality
        )
    }

    fun takeScreenshot(screenshotName: String?) {
        device.takeScreenshot(
            File(getScreenshotDevicePath(), childId = "$screenshotName.png"),
            screenshotScale,
            screenshotQuality
        )
    }
}
```

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.

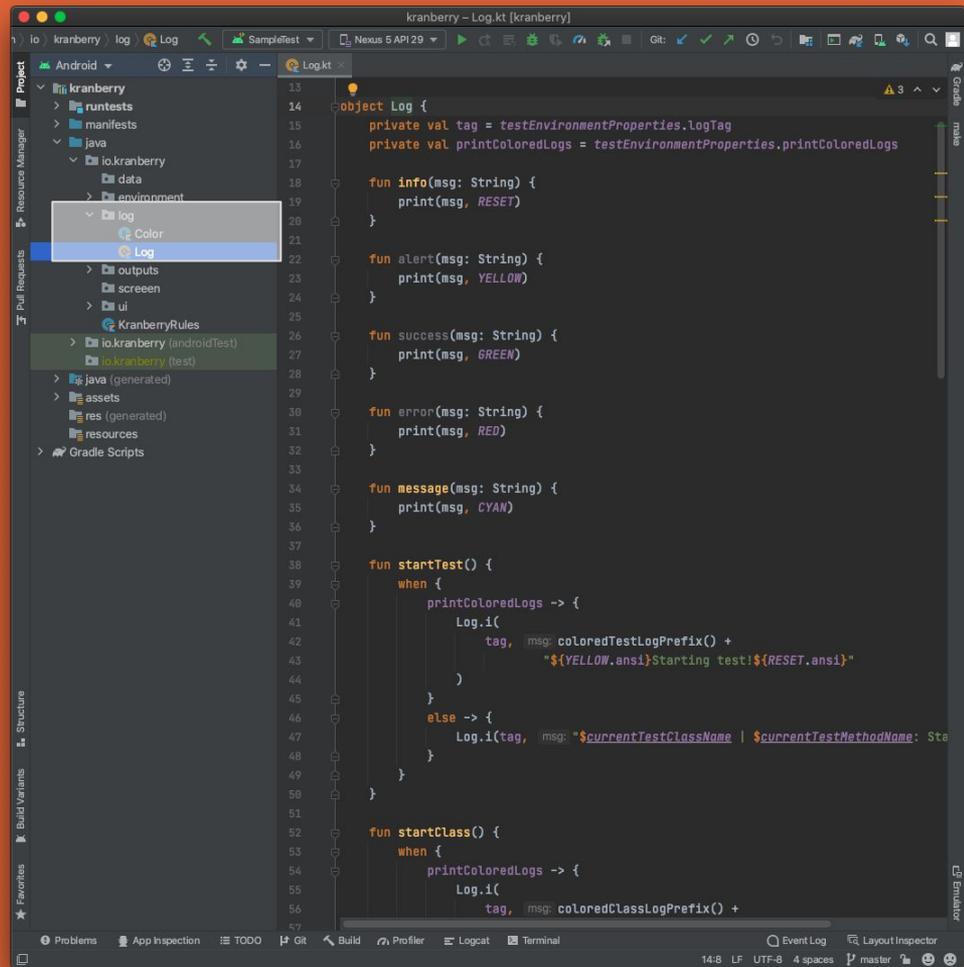


```
1 package io.kranberry
2
3 import ..
4
5
6
7
8 interface RunTestsPluginExtension {
9     val packageName: Property<String>
10    val packageTest: Property<String>
11    val applicationId: Property<String>
12    val apkOutputPath: Property<String>
13    val androidTestApkOutputPath: Property<String>
14    val testsRunner: Property<String>
15 }
16
17 class RunTestsPlugin : Plugin<Project> {
18     @ExperimentalStdLibApi
19     override fun apply(project: Project) {
20         val extension = project.extensions.create(name = "kranberryTests", RunTestsPL
21         extension.run { this: RunTestsPluginExtension
22
23             val assemble = project.tasks.findByName(name = "assemble")
24             val assembleAndroidTest = project.tasks.findByName(name = "assembleAndroi
25
26             val getKranberryTestsResults = project.task(name = "getKranberryTestsResu
27             task.doLast { it: Task
28                 project.exec { it: ExecSpec
29                     // Pull results
30                     val resultsPackage = applicationId.getOrElse(packageName.get
31                     val pullPath = "/storage/emulated/0/Android/media/$resultsPa
32                     it.commandLine(...args: "bash", "-L", "-c", "rm -rf kranberry-c
33                     it.commandLine(...args: "bash", "-L", "-c", "mkdir kranberry-ol
34                     it.commandLine(...args: "bash", "-L", "-c", "adb pull $pullPath
35                 }
36             }
37
38             val installKranberryAps = project.task(name = "installKranberryAps") {
39                 task.doFirst { it: Task
40                     project.exec { it: ExecSpec
41                         // Install App APK
42                         if (apkOutputPath.isPresent) {
43                             println("Install APK: ${apkOutputPath.get()}")
44                             it.commandLine(...args:
45                                 "bash", "-L", "-c",
46                                 "pwd"
47                         )
48                     }
49                 }
50             }
51         }
52     }
53 }
```

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.



```
object Log {
    private val tag = testEnvironmentProperties.logTag
    private val printColoredLogs = testEnvironmentProperties.printColoredLogs

    fun info(msg: String) {
        print(msg, RESET)
    }

    fun alert(msg: String) {
        print(msg, YELLOW)
    }

    fun success(msg: String) {
        print(msg, GREEN)
    }

    fun error(msg: String) {
        print(msg, RED)
    }

    fun message(msg: String) {
        print(msg, CYAN)
    }

    fun startTest() {
        when {
            printColoredLogs -> {
                Log.i(
                    tag, msg: coloredTestLogPrefix() +
                        "${YELLOW.ansi}Starting test!${RESET.ansi}"
                )
            }
            else -> {
                Log.i(tag, msg: "$currentTestClass | $currentTestMethod: Sta
            }
        }
    }

    fun startClass() {
        when {
            printColoredLogs -> {
                Log.i(
                    tag, msg: coloredClassLogPrefix() +

```

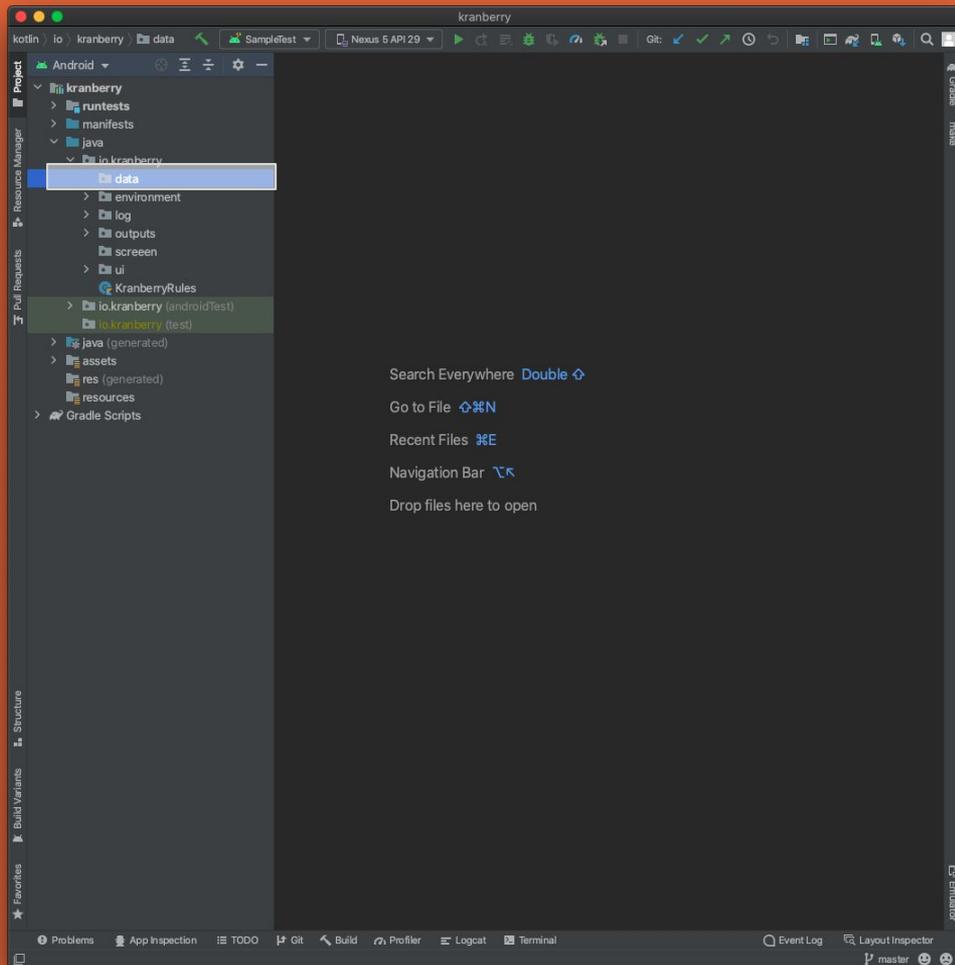
Lidar com testes guiados por dados externos

package io.kranberry.data



Está no Roadmap!

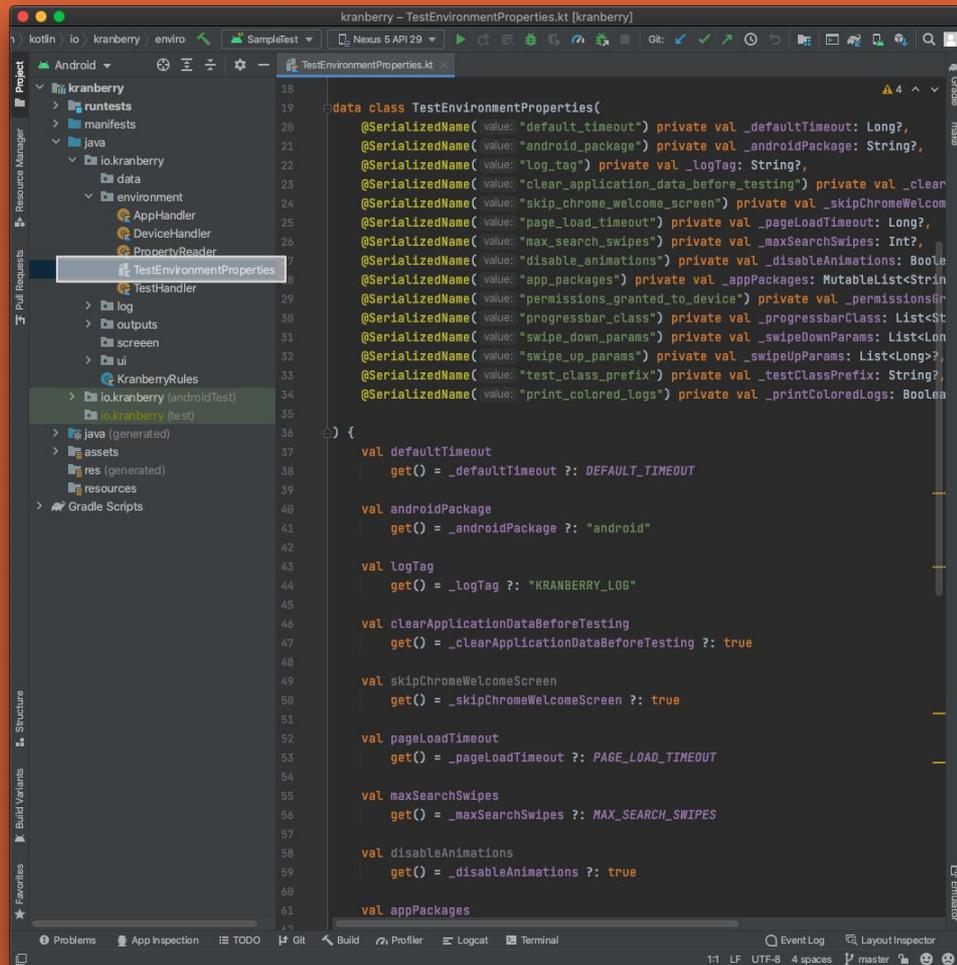
Ainda sim, já temos uma estrutura de leitura de arquivos externos, precisamos agora concluir a implementação!



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.

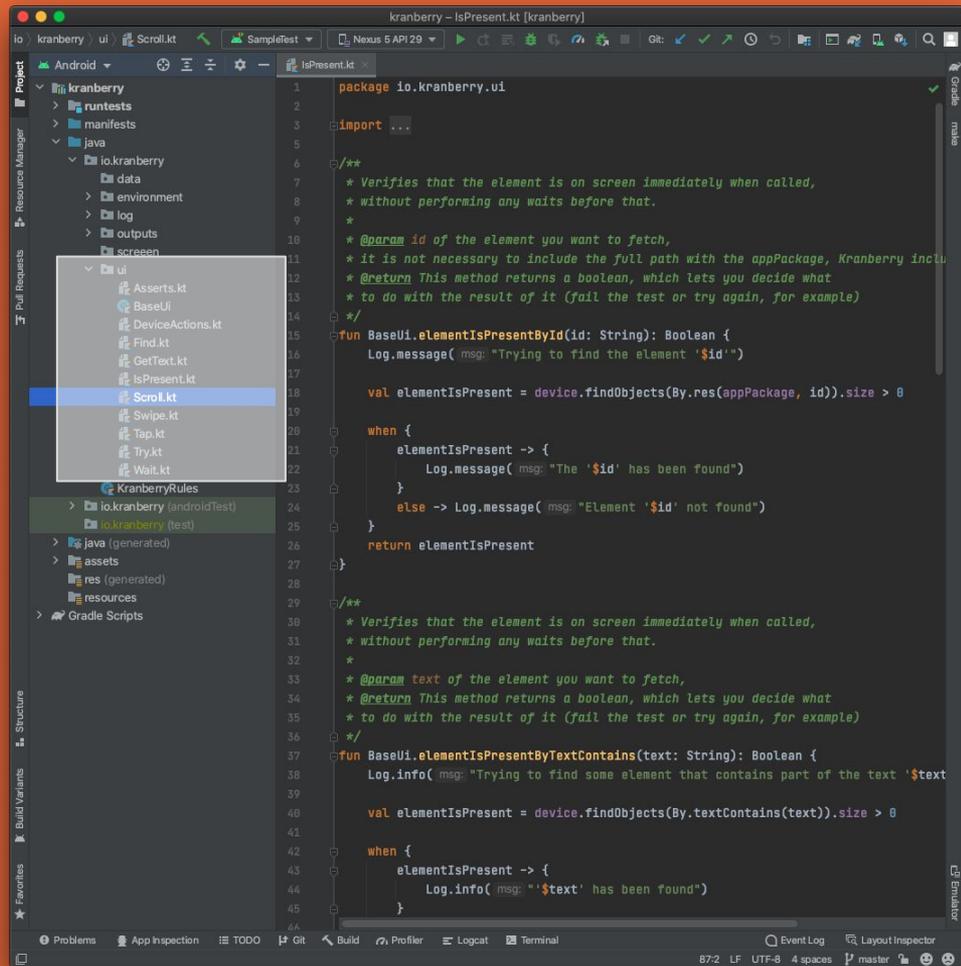


```
18
19
20 data class TestEnvironmentProperties(
21     @SerializedName(value = "default_timeout") private val _defaultTimeout: Long?,
22     @SerializedName(value = "android_package") private val _androidPackage: String?,
23     @SerializedName(value = "log_tag") private val _logTag: String?,
24     @SerializedName(value = "clear_application_data_before_testing") private val _clear
25     @SerializedName(value = "skip_chrome_welcome_screen") private val _skipChromeWelcom
26     @SerializedName(value = "page_load_timeout") private val _pageLoadTimeout: Long?,
27     @SerializedName(value = "max_search_swipes") private val _maxSearchSwipes: Int?,
28     @SerializedName(value = "disable_animations") private val _disableAnimations: Boolean
29     @SerializedName(value = "app_packages") private val _appPackages: MutableList<Strin
30     @SerializedName(value = "permissions_granted_to_device") private val _permissionsGr
31     @SerializedName(value = "progressbar_class") private val _progressbarClass: List<St
32     @SerializedName(value = "swipe_down_params") private val _swipeDownParams: List<Lon
33     @SerializedName(value = "swipe_up_params") private val _swipeUpParams: List<Long?
34     @SerializedName(value = "test_class_prefix") private val _testClassPrefix: String?
35     @SerializedName(value = "print_colored_logs") private val _printColoredLogs: Boolean
36 ) {
37     val defaultTimeout
38         get() = _defaultTimeout ?: DEFAULT_TIMEOUT
39
40     val androidPackage
41         get() = _androidPackage ?: "android"
42
43     val logTag
44         get() = _logTag ?: "KRANBERRY_LOG"
45
46     val clearApplicationDataBeforeTesting
47         get() = _clearApplicationDataBeforeTesting ?: true
48
49     val skipChromeWelcomeScreen
50         get() = _skipChromeWelcomeScreen ?: true
51
52     val pageLoadTimeout
53         get() = _pageLoadTimeout ?: PAGE_LOAD_TIMEOUT
54
55     val maxSearchSwipes
56         get() = _maxSearchSwipes ?: MAX_SEARCH_SWIPES
57
58     val disableAnimations
59         get() = _disableAnimations ?: true
60
61     val appPackages
```

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.



```
package io.kranberry.ui

import ...

/**
 * Verifies that the element is on screen immediately when called,
 * without performing any waits before that.
 *
 * @param id of the element you want to fetch,
 * It is not necessary to include the full path with the appPackage, Kranberry includes it
 * @return This method returns a boolean, which lets you decide what
 * to do with the result of it (fail the test or try again, for example)
 */
fun BaseUi.elementIsPresentById(id: String): Boolean {
    Log.message(msg: "Trying to find the element '$id'")

    val elementIsPresent = device.findObjects(By.res(appPackage, id)).size > 0

    when {
        elementIsPresent -> {
            Log.message(msg: "The '$id' has been found")
        }
        else -> Log.message(msg: "Element '$id' not found")
    }

    return elementIsPresent
}

/**
 * Verifies that the element is on screen immediately when called,
 * without performing any waits before that.
 *
 * @param text of the element you want to fetch,
 * @return This method returns a boolean, which lets you decide what
 * to do with the result of it (fail the test or try again, for example)
 */
fun BaseUi.elementIsPresentByTextContains(text: String): Boolean {
    Log.info(msg: "Trying to find some element that contains part of the text '$text'")

    val elementIsPresent = device.findObjects(By.textContains(text)).size > 0

    when {
        elementIsPresent -> {
            Log.info(msg: "'$text' has been found")
        }
    }
}
```

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
Kranberry, você é
capaz de
configurar seu
pacote de testes
E2E em minutos!**





```

kranberry-sample app build.gradle
Project
  Project
    kranberry-sample [KranberrySample] ~/Documents/kranberry-sample
      .google
      .gradle
      .idea
      app
        build
        src
        .gitignore
        build.gradle
        build
        gradle
        .gitignore
        build.gradle
        gradle.properties
        gradlew
        gradlew.bat
        LICENSE
        local.properties
        README.md
        settings.gradle
      External Libraries
      Scratches and Consoles
  Pull Requests
  Resource Manager
  Structure
  Favorites
  Build Variants
dependencies()
54 implementation org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version
55 implementation androidx.appcompat:appcompat:$rootProject.appCompatVersion
56 implementation androidx.core:core-ktx:$rootProject.coreKtxVersion
57 implementation androidx.constraintlayout:constraintlayout:$rootProject.constraintLayoutVersion
58 implementation androidx.activity:activity-ktx:$rootProject.activityVersion
59
60 // RecyclerView
61 implementation androidx.recyclerview:recyclerview:$rootProject.recyclerviewVersion
62
63 // Material design
64 implementation com.google.android.material:material:$rootProject.materialVersion
65
66 // LiveData
67 implementation androidx.lifecycle:lifecycle-livedata-ktx:$rootProject.liveDataVersion
68
69 // Kranberry
70 androidTestImplementation 'io.github.kranberry-io:kranberry:1.0.1-beta'
71
72
    
```

Terminal: Local +

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.

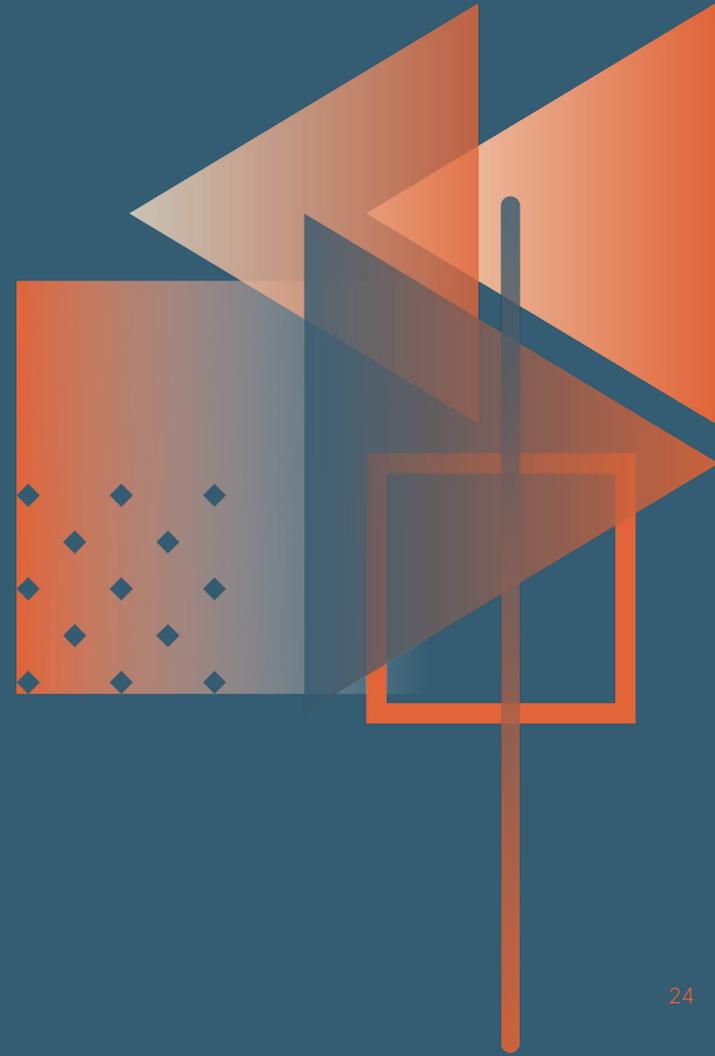
```

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

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

00:01:32

E o futuro?



Novas funcionalidades + estabilidade

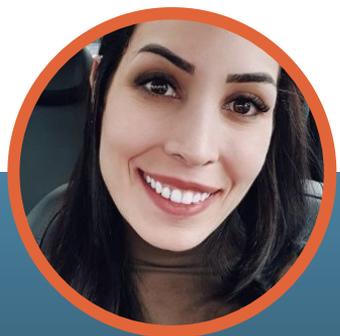
Roadmap, idéias...

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?**