Танцы вокруг дерева семантики: UI-тесты Compose-экранов

В 2024 году
Jetpack Compose
Никого не удивляет

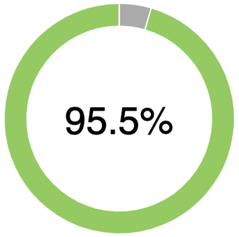

АДАПТАЦИЯ
COMPOSE В hh
https://www.youtube.com/watch?v=_XJnMs_nKZY

ALLURE REPORT 8/21/2024

1:28:05 - 1:52:44 (24m 38s)

512

test cases



SUITES 509 items total

ru.hh.shared.core.matchers.tests.QueryUrlsMatcherTest	3
ru.hh.android.integration.resume.ResumeContactsTest	1 / 1
ru.hh.android.integration.resume.ArtifactConditionsApiTest	1
ru.hh.android.test.vacancy.complaints.ComplaintVacancyWithReportTest	1
ru.hh.android.test.vacancy.suitable.EmptyRecommendedVacancyBlockTest	1
ru.hh.android.test.vacancy.suitable.ManySuitableVacanciesInRecommendedBlockTest	1
ru.hh.android.test.vacancy.complaints.ComplaintVacancyWithOpenArticleLinkTest	1
ru.hh.android.test.vacancy.suitable.SingleSuitableVacancyInRecommendedBlockTest	1
ru.hh.android.test.vacancy.YesterdayVacancyWithAddressAndContactsScreenTest	1
ru.hh.android.test.vacancy.CheckGoToChatButtonDoesNotExistsAfterLeaveChatTest	1

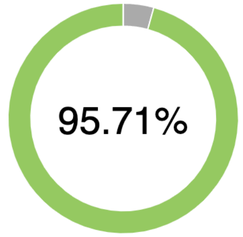
Show all

ALLURE REPORT 8/21/2024

1:28:08 - 1:40:37 (12m 28s)

257

test cases

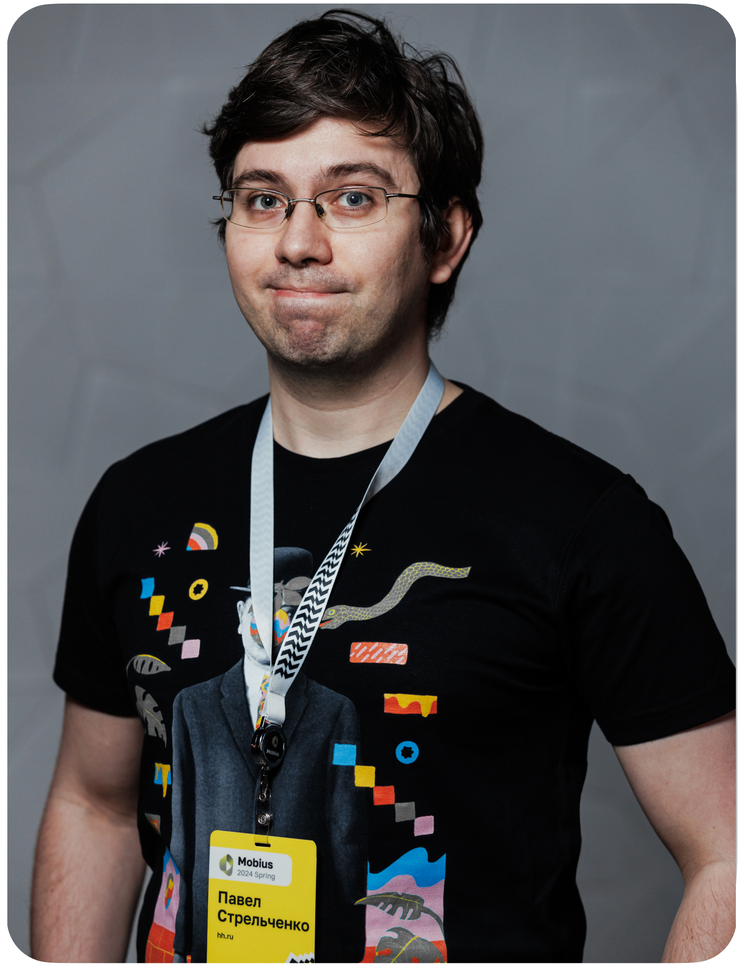


SUITES 255 items total

ru.hh.shared.core.matchers.tests.QueryUrlsMatcherTest	3
ru.hh.android.hr_mobile.ui_test.test.auth.AuthEmployerWith2FaTest	1
ru.hh.android.hr_mobile.ui_test.test.vacancy_builder.steps_tests.WhereApplicantWillWorkStepTest	1
ru.hh.android.hr_mobile.ui_test.test.vacancy_builder.screens_tests.WhatScheduleScreenTest	1
ru.hh.android.hr_mobile.ui_test.test.vacancy_builder.steps_tests.WhatPersonIsSuitableVacancyStepTest	1
ru.hh.android.hr_mobile.ui_test.test.vacancy_builder.screens_tests.WhereApplicantWillWorkScreenTest	1
ru.hh.android.hr_mobile.ui_test.test.vacancy_builder.steps_tests.VacancyDescriptionErrorStepTest	1
ru.hh.android.hr_mobile.ui_test.test.vacancy_builder.steps_tests.AmountPaymentStepTest	1
ru.hh.android.hr_mobile.ui_test.test.vacancy_builder.screens_tests.WhoAreYouLookingScreenTest	1
ru.hh.android.hr_mobile.ui_test.test.vacancy_builder.steps_tests.AdditionalContactsStepTest	1

Show all

Давайте познакомимся



Павел Стрельченко

Тимлид команды мобильной дизайн-системы в hh.ru

Меня зовут Android-разработчик, я работаю Пашей (с)

Давно работаю в платформенных командах hh.ru

Помогаю коллегам решать их задачи

@Ztrel

@PStrelchenko

Что вас сегодня ждёт?

Что есть из коробки?

Расскажу про стандартный фреймворк тестирования Compose-экранов от Google

Улучшаем API

Покажу, как мы итеративно улучшали API UI-тестов в нашей кодовой базе на базе Kaspresso

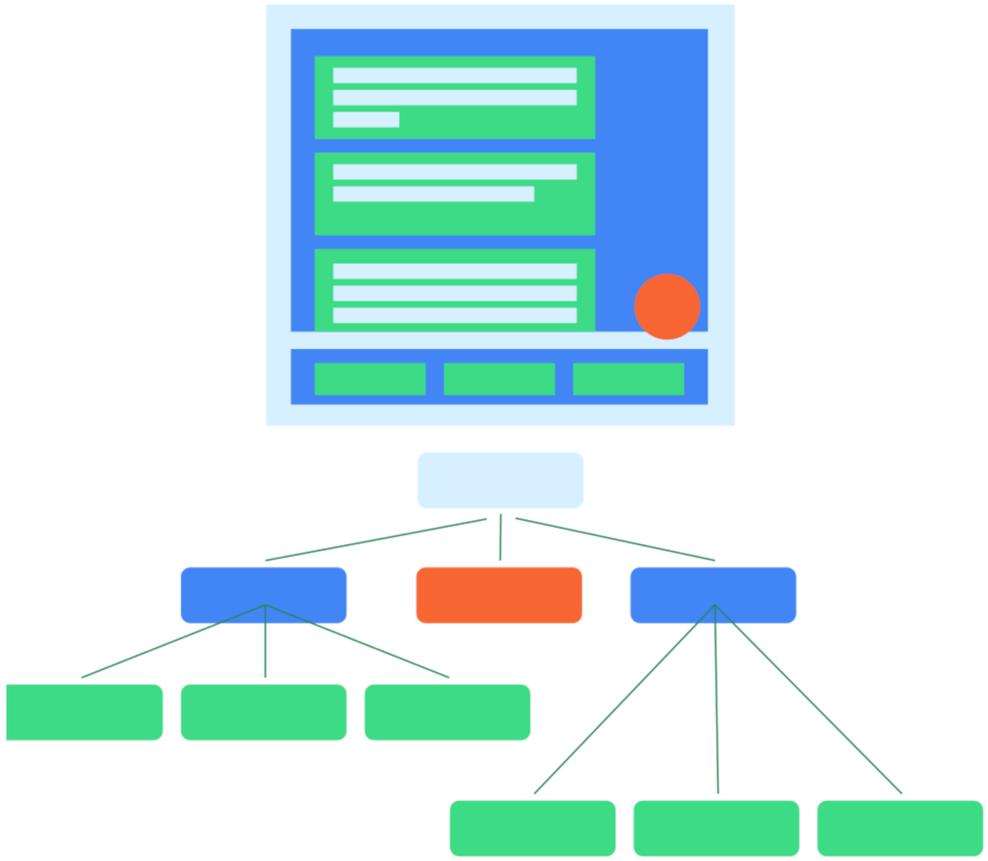
Tips & tricks

Обсудим подводные камни в тестировании Compose-экранов и как их обойти

Акт I

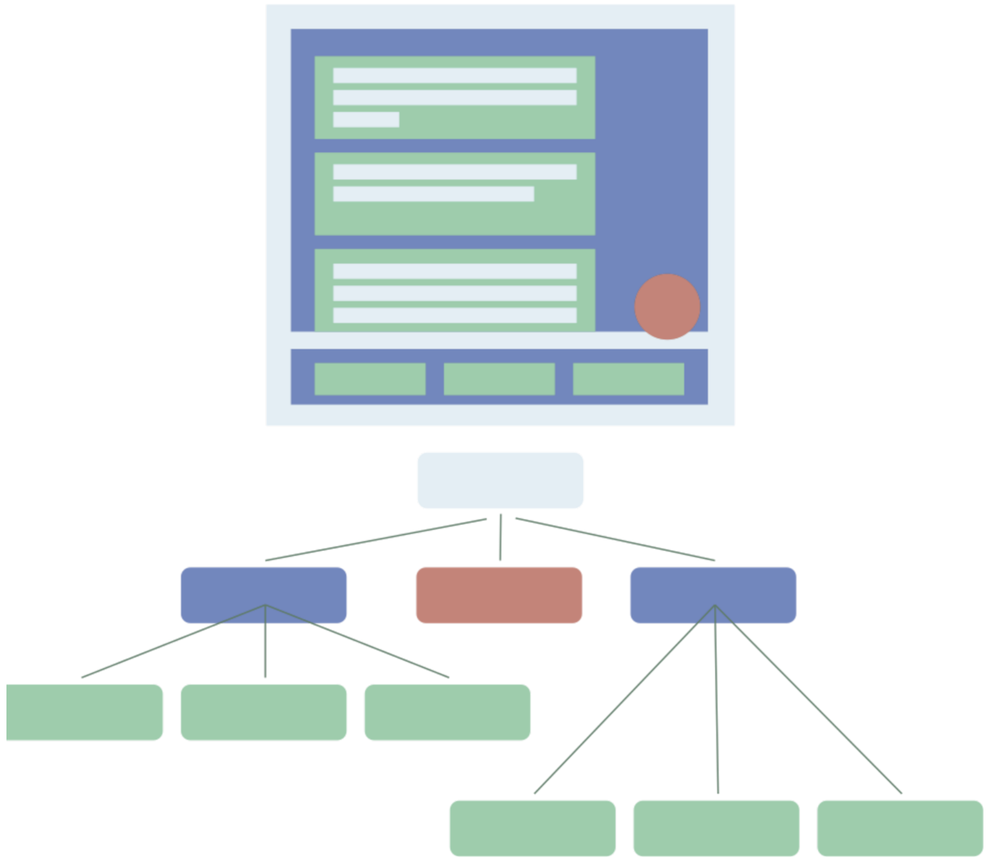
Что доступно «из коробки»?

Работаем с семантическим деревом



```
Column(
    modifier = Modifier
        .verticalScroll(rememberScrollState())
) {
    Button(
        modifier = Modifier
            .clickable { /* do smth */ }
    ) {
        Text(
            text = "Click me!",
            modifier = Modifier.semantics {
                testTag = "screen::buttonText"
            }
        )
    }
}
```

Modifier и виджеты добавляют семантику в узлы



```
Column(
    modifier = Modifier
        .verticalScroll(rememberScrollState())
) {
    Button(
        modifier = Modifier
            .clickable { /* do smth */ }
    ) {
        Text(
            text = "Click me!",
            modifier = Modifier.semantics {
                testTag = "screen::buttonText"
            }
        )
    }
}
```

Семантические свойства можно добавлять

```
/**
 * Идентификатор картинки, которую установили в компонент Icon / Image.
 */
val IconRes = SemanticsPropertyKey<Int>("IconRes")

/**
 * Позволяет использовать getter и setter [IconRes]
 * внутри блока [Modifier.semantics].
 */
@get:DrawableRes
var SemanticsPropertyReceiver.iconRes by IconRes
```

Дерево виджетов и дерево семантики

Не равны друг другу

Почему деревья не равны?

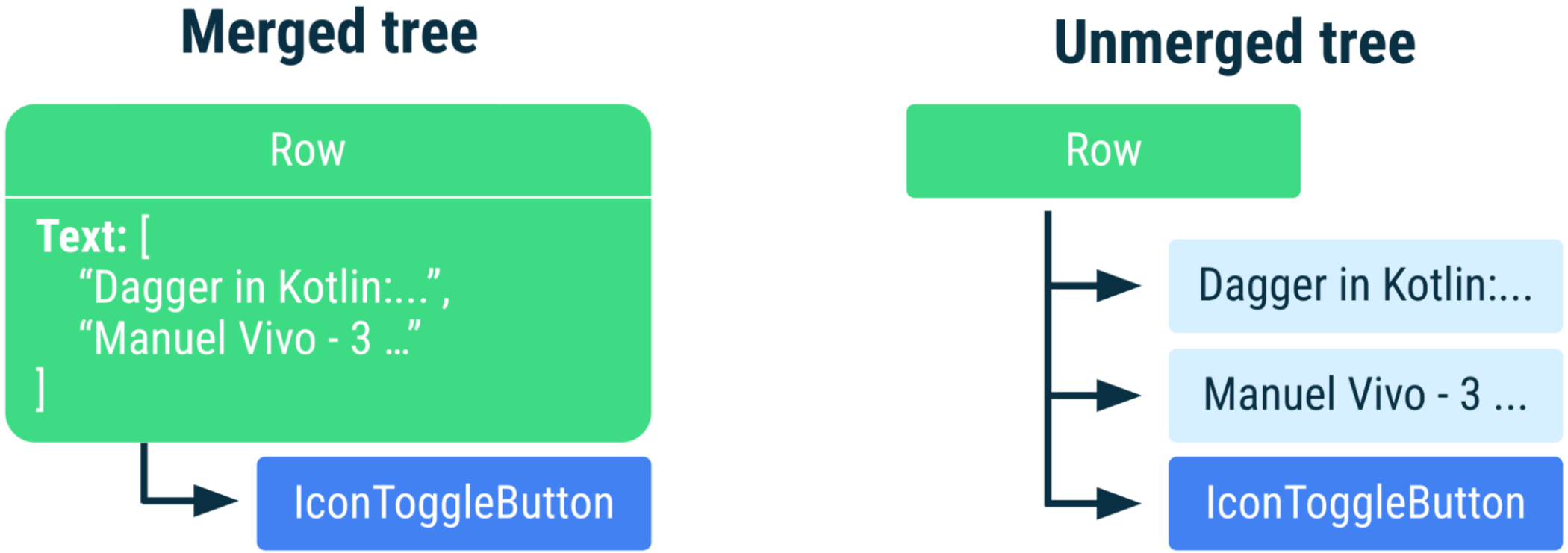
Узлы без семантики

Если виджет не добавляет семантику — он отсутствует в дереве семантики

Merged vs Unmerged

В merged дереве виджеты соединяют свою семантику, образуя изменённые узлы

Merged vs Unmerged



Google предоставляет jUnit Rule

```
abstract class MagritteGalleryTestCase(/* ... */) {

    // Специальный rule для Compose-тестов.
    // Не указываем Activity, чтобы она не запускалась этим правилом
    @get:Rule
    val composeTestRule = createEmptyComposeRule()

}
```

ComposeTestRule для работы с деревом

```
class SomeTestCase(/* ... */) : MagritteGalleryTestCase() {

    @Test
    fun myTest() {
        composeTestRule.onNodeWithText("Continue")
            .performClick()

        composeTestRule.onNodeWithTag("screen::buttonText")
            .assertIsDisplayed()

        composeTestRule.onNode(
            SemanticsMatcher.expectValue(
                SemanticsProperties.Role, Role.RadioButton
            ).and(
                SemanticsMatcher.keyIsDefined(SemanticsProperties.Selected)
            )
        ).printToLog("LOG_TAG")
    }

}
```

Сначала описываем поиск элемента...

```
class SomeTestCase(/* ... */) : MagritteGalleryTestCase() {

    @Test
    fun myTest() {
        composeTestRule.onNodeWithText("Continue")
            .performClick()

        composeTestRule.onNodeWithTag("screen::buttonText")
            .assertIsDisplayed()

        composeTestRule.onNode(
            SemanticsMatcher.expectValue(
                SemanticsProperties.Role, Role.RadioButton
            ).and(
                SemanticsMatcher.keyIsDefined(SemanticsProperties.Selected)
            )
        ).printToLog("LOG_TAG")
    }

}
```

Ищем либо по тестовому тегу (аналог @+id/)

```
class SomeTestCase(/* ... */) : MagritteGalleryTestCase() {

    @Test
    fun myTest() {
        composeTestRule.onNodeWithText("Continue")
            .performClick()

        composeTestRule.onNodeWithTag("screen::buttonText")
            .assertIsDisplayed()

        composeTestRule.onNode(
            SemanticsMatcher.expectValue(
                SemanticsProperties.Role, Role.RadioButton
            ).and(
                SemanticsMatcher.keyIsDefined(SemanticsProperties.Selected)
            )
        ).printToLog("LOG_TAG")
    }

}
```

Либо по другим семантическим свойствам

```
class SomeTestCase(/* ... */) : MagritteGalleryTestCase() {

    @Test
    fun myTest() {
        composeTestRule.onNodeWithText("Continue")
            .performClick()

        composeTestRule.onNodeWithTag("screen::buttonText")
            .assertIsDisplayed()

        composeTestRule.onNode(
            SemanticsMatcher.expectValue(
                SemanticsProperties.Role, Role.RadioButton
            ).and(
                SemanticsMatcher.keyIsDefined(SemanticsProperties.Selected)
            )
        ).printToLog("LOG_TAG")
    }

}
```

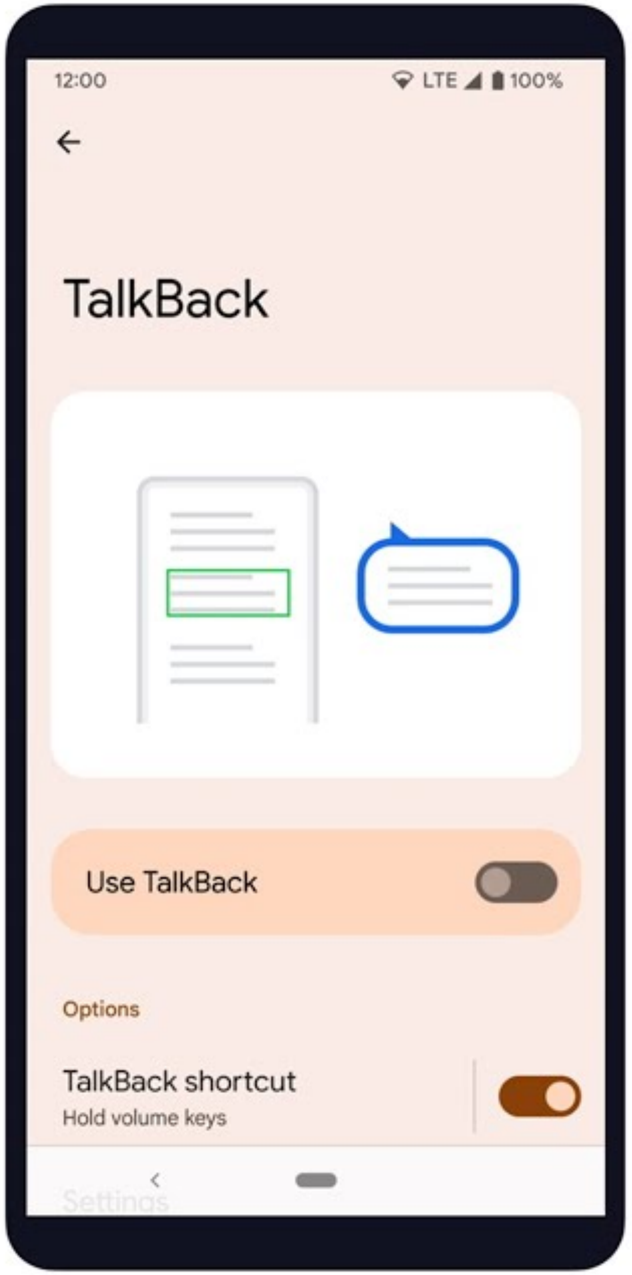
... а потом выполняем проверки или действия

```
class SomeTestCase(/* ... */) : MagritteGalleryTestCase() {

    @Test
    fun myTest() {
        composeTestRule.onNodeWithText("Continue")
            .performClick()

        composeTestRule.onNodeWithTag("screen::buttonText")
            .assertIsDisplayed()

        composeTestRule.onNode(
            SemanticsMatcher.expectValue(
                SemanticsProperties.Role, Role.RadioButton
            ).and(
                SemanticsMatcher.keyIsDefined(SemanticsProperties.Selected)
            )
        ).printToLog("LOG_TAG")
    }

}
```

По умолчанию,
мы работаем с
merged-деревом

Android Accessibility

TalkBack



https://www.youtube.com/watch?v=_1yRVwhEv5I

Стандартное API — гибкое, однако...

1. **Не предоставляется структуры тестов**
2. Нет встроенной возможности переиспользования кода
3. Нет защиты от флакования

Стандартное API — гибкое, однако...

1. Не предоставляется структуры тестов
2. **Нет встроенной возможности переиспользования кода**
3. Нет защиты от флакования

Стандартное API — гибкое, однако...

1. Не предоставляется структуры тестов
2. Нет встроенной возможности переиспользования кода
3. **Нет защиты от флакования**

А что предлагает open source?

Как минимум — два фреймворка

Ultron

https://github.com/open-tool/ultron

Kaspresso

https://github.com/KasperskyLab/Kaspresso/

В hh.ru мы давно используем Kaspresso

Ultron

https://github.com/open-tool/ultron

Kaspresso

https://github.com/KasperskyLab/Kaspresso/

Что нам даёт Kaspresso?

1. **Структура кода тестов — Page Object, блоки init, before, etc**
2. Расширенные возможности тестирования (e.g. ADB в тестах)
3. Allure-отчёты с шагами
4. Защита от флакования в UI-тестах
5. И ещё много всего интересного

Что нам даёт Kaspresso?

1. Структура кода тестов — Page Object, блоки init, before, etc
2. **Расширенные возможности тестирования (e.g. ADB в тестах)**
3. Allure-отчёты с шагами
4. Защита от флакования в UI-тестах
5. И ещё много всего интересного

Что нам даёт Kaspresso?

1. Структура кода тестов — Page Object, блоки init, before, etc
2. Расширенные возможности тестирования (e.g. ADB в тестах)
3. **Allure-отчёты с шагами**
4. Защита от флакования в UI-тестах
5. И ещё много всего интересного

Что нам даёт Kaspresso?

1. Структура кода тестов — Page Object, блоки init, before, etc
2. Расширенные возможности тестирования (e.g. ADB в тестах)
3. Allure-отчёты с шагами
4. **Защита от флакования в UI-тестах**
5. И ещё много всего интересного

Что нам даёт Kaspresso?

1. Структура кода тестов — Page Object, блоки init, before, etc
2. Расширенные возможности тестирования (e.g. ADB в тестах)
3. Allure-отчёты с шагами
4. Защита от флакования в UI-тестах
5. **И ещё много всего интересного**

Kaspresso оборачивает библиотеку Kakao

А что нам даёт Kakao?

1. **Kotlin DSL для взаимодействия с элементами**
2. Возможность добавлять interceptor для каждого действия или проверки с элементами
3. Базовые классы для работы с узлами семантического дерева, например, KNode, KIconNode, KLazyListNode, etc

А что нам даёт Kakao?

1. Kotlin DSL для взаимодействия с элементами
2. **Возможность добавлять interceptor для каждого действия или проверки с элементами**
3. Базовые классы для работы с узлами семантического дерева, например, KNode, KIconNode, KLazyListNode, etc

А что нам даёт Kakao?

1. Kotlin DSL для взаимодействия с элементами
2. Возможность добавлять interceptor для каждого действия или проверки с элементами
3. **Базовые классы для работы с узлами семантического дерева, например, KNode, KIconNode, KLazyListNode, etc**

Как выглядят Page Object в нашем коде

```
internal class AboutComposePageObject(
    semanticsProvider: SemanticsNodeInteractionsProvider,
) : ComposePageObject<AboutComposePageObject>(
    semanticsProvider = semanticsProvider,
) {

    val actions = Actions()
    val checks = Checks()

    // region Внутренние элементы
    private val header = KNode(AboutScreenTestTags.header)
    // ...

    // endregion

    inner class Actions : PageObjectIntentions<Actions>() {
        // ...
    }

    inner class Checks : PageObjectIntentions<Checks>() {
        // ...
    }
```

Наследуемся от ComposePageObject

```
internal class AboutComposePageObject(
    semanticsProvider: SemanticsNodeInteractionsProvider,
) : ComposePageObject<AboutComposePageObject>(
    semanticsProvider = semanticsProvider,
) {

    val actions = Actions()
    val checks = Checks()

    // region Внутренние элементы
    private val header = KNode(AboutScreenTestTags.header)
    // ...

    // endregion

    inner class Actions : PageObjectIntentions<Actions>() {
        // ...
    }

    inner class Checks : PageObjectIntentions<Checks>() {
        // ...
    }
```

```
open class ComposePageObject<out T : ComposePageObject<T>>
    : BaseNode<T>
```

Описываем поиск внутренних элементов экрана

```
internal class AboutComposePageObject(
    semanticsProvider: SemanticsNodeInteractionsProvider,
) : ComposePageObject<AboutComposePageObject>(
    semanticsProvider = semanticsProvider,
) {

    val actions = Actions()
    val checks = Checks()

    // region Внутренние элементы
    private val header = KNode(AboutScreenTestTags.header)
    // ...

    // endregion

    inner class Actions : PageObjectIntentions<Actions>() {
        // ...
    }

    inner class Checks : PageObjectIntentions<Checks>() {
        // ...
    }
```

Добавляем описание действий и проверок

```
internal class AboutComposePageObject(
    semanticsProvider: SemanticsNodeInteractionsProvider,
) : ComposePageObject<AboutComposePageObject>(
    semanticsProvider = semanticsProvider,
) {

    val actions = Actions()
    val checks = Checks()

    // region Внутренние элементы
    private val header = KNode(AboutScreenTestTags.header)
    // ...

    // endregion

    inner class Actions : PageObjectIntentions<Actions>() {
        // ...
    }

    inner class Checks : PageObjectIntentions<Checks>() {
        // ...
    }
```

Классы Actions / Checks

```
internal class AboutComposePageObject(
    semanticsProvider: SemanticsNodeInteractionsProvider,
) : ComposePageObject<AboutComposePageObject>(
    semanticsProvider = semanticsProvider,
) {
    // ...

    inner class Actions : PageObjectIntentions<Actions>() {

        fun TestContext<*>.openAnotherScreen() {
            step("Открываем экран Another") {
                mainButton.performClick()
            }
        }

    }

    // ...

}
```

Типичный тест на Kaspresso

```
internal class BrandedButtonScreenDefaultStateTest : MagritteGalleryTestCase() {

    private val mainPageObject = MainPageObject(composeTestRule)
    private val brandedButtonPageObject = BrandedButtonPageObject(composeTestRule)

    @Test
    fun test() = run {
        mainPageObject.actions { openBrandedButtonScreen() }
        brandedButtonPageObject.checks { checkScreenTitle("BrandedButton") }

        step("Проверяем начальное состояние кнопки с единичным сервисом") {
            brandedButtonPageObject.checks {
                checkBrandedButtonIsCorrect(
                    KBrandedButtonCheckParams(
                        firstIconIsDisplayed = true,
                        firstIcon = MT.icon.googleBrand24,
                        firstIconHasTint = false,
                    )
                )
            }
    }
```

Наследуемся от базового класса с набором Rule

```
internal class BrandedButtonScreenDefaultStateTest : MagritteGalleryTestCase() {

    private val mainPageObject = MainPageObject(composeTestRule)
    private val brandedButtonPageObject = BrandedButtonPageObject(composeTestRule)

    @Test
    fun test() = run {
        mainPageObject.actions { openBrandedButtonScreen() }
        brandedButtonPageObject.checks { checkScreenTitle("BrandedButton") }

        step("Проверяем начальное состояние кнопки с единичным сервисом") {
            brandedButtonPageObject.checks {
                checkBrandedButtonIsCorrect(
                    KBrandedButtonCheckParams(
                        firstIconIsDisplayed = true,
                        firstIcon = MT.icon.googleBrand24,
                        firstIconHasTint = false,
                    )
                )
            }
    }
```

Перечисляем необходимые Page Object

```
internal class BrandedButtonScreenDefaultStateTest : MagritteGalleryTestCase() {

    private val mainPageObject = MainPageObject(composeTestRule)
    private val brandedButtonPageObject = BrandedButtonPageObject(composeTestRule)

    @Test
    fun test() = run {
        mainPageObject.actions { openBrandedButtonScreen() }
        brandedButtonPageObject.checks { checkScreenTitle("BrandedButton") }

        step("Проверяем начальное состояние кнопки с единичным сервисом") {
            brandedButtonPageObject.checks {
                checkBrandedButtonIsCorrect(
                    KBrandedButtonCheckParams(
                        firstIconIsDisplayed = true,
                        firstIcon = MT.icon.googleBrand24,
                        firstIconHasTint = false,
                    )
                )
            }
        }
    }
```

Описываем логику теста

```
internal class BrandedButtonScreenDefaultStateTest : MagritteGalleryTestCase() {

    private val mainPageObject = MainPageObject(composeTestRule)
    private val brandedButtonPageObject = BrandedButtonPageObject(composeTestRule)

    @Test
    fun test() = run {
        mainPageObject.actions { openBrandedButtonScreen() }
        brandedButtonPageObject.checks { checkScreenTitle("BrandedButton") }

        step("Проверяем начальное состояние кнопки с единичным сервисом") {
            brandedButtonPageObject.checks {
                checkBrandedButtonIsCorrect(
                    KBrandedButtonCheckParams(
                        firstIconIsDisplayed = true,
                        firstIcon = MT.icon.googleBrand24,
                        firstIconHasTint = false,
                    )
                )
            }
        }
    }
```

Подведём промежуточные итоги

1. **Для UI-тестов используется дерево семантики**
2. Стандартное API тестирования не форсирует хорошие практики и не защищает от флакования
3. В hh.ru в качестве основного фреймворка используется Kaspresso — поэтому речь дальше пойдёт про него

Подведём промежуточные итоги

1. Для UI-тестов используется дерево семантики
2. **Стандартное API тестирования не форсирует хорошие практики и не защищает от флакования**
3. В hh.ru в качестве основного фреймворка используется Kaspresso — поэтому речь дальше пойдёт про него

Подведём промежуточные итоги

1. Для UI-тестов используется дерево семантики
2. Стандартное API тестирования не форсирует хорошие практики и не защищает от флакования
3. **В hh.ru в качестве основного фреймворка используется Kaspresso — поэтому речь дальше пойдёт про него**

Акт II

Улучшаем API тестов

Какие видим проблемы в существующем коде

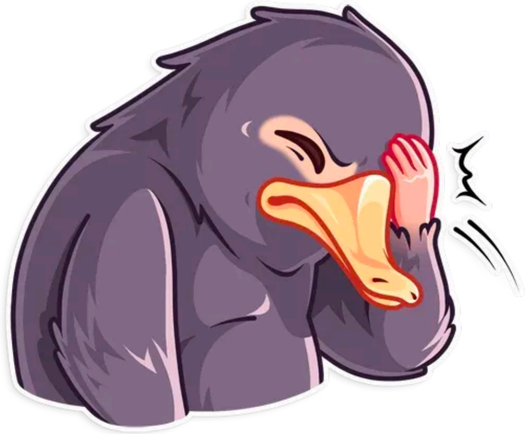
1. Нужно везде пробрасывать ComposeTestRule

ComposeTestRule нужно пробрасывать вглубь

```
class MyComposePageObject(
    semanticsProvider: SemanticsNodeInteractionsProvider
): ComposePageObject<MyComposePageObject>(semanticsProvider) {

    private val anotherPageObject = AnotherPageObject(semanticsProvider)
    private val oneMorePageObject = OneMorePageObject(semanticsProvider)

}
```

ComposeTestRule нужно пробрасывать вглубь

```
class MyComposePageObject(
    semanticsProvider: SemanticsNodeInteractionsProvider
): ComposePageObject<MyComposePageObject>(semanticsProvider) {

    private val anotherPageObject = AnotherPageObject(semanticsProvider)
    private val oneMorePageObject = OneMorePageObject(semanticsProvider)

}
```



Можно ли где-то сохранить ComposeTestRule?...

Можно ввести контейнер для ComposeTestRule

```
object ComposeRuleContainer {

    var rule: ComposeTestRule? = null

    fun init(composeRule: ComposeTestRule) {
        rule = composeRule
        testContext = composeRule.getTestContext()
    }

    fun getComposeRule(): ComposeTestRule {
        return requireNotNull(rule) {
            "Initialise ComposeTestRule using
            ComposeRuleFactory.createEmptyMagritteComposeRule"
        }
    }
}
```

Сохраняем ComposeTestRule при создании

```
object ComposeRuleFactory {

    fun createEmptyMagritteComposeRule(): ComposeTestRule {
        val rule = createEmptyComposeRule()

        ComposeRuleContainer.init(rule)

        return rule
    }

}
```

Это убирает кучу дополнительного кода

```
class MyComposePageObject(): ComposePageObject<MyComposePageObject>() {

    private val anotherPageObject = AnotherPageObject()
    private val oneMorePageObject = OneMorePageObject()

}
```

Более корректно — создать свой TestRule

```
class KakaoComposeTestRule(
    val newComposeTestRule: ComposeTestRule? = null
) : TestRule {
    override fun apply(base: Statement, description: Description): Statement =
        object : Statement() {
            override fun evaluate() {
                val previousComposeRule = ComposeRuleContainer.savedComposeRule
                newComposeTestRule?.let {
                    ComposeRuleContainer.savedComposeRule = it
                }

                base.evaluate()

                ComposeRuleContainer.savedComposeRule = previousComposeRule
            }
        }
}
```

Какие видим проблемы в существующем коде

1. Нужно везде пробрасывать ComposeTestRule

→ Ввели контейнер для хранения ComposeTestRule между тестами

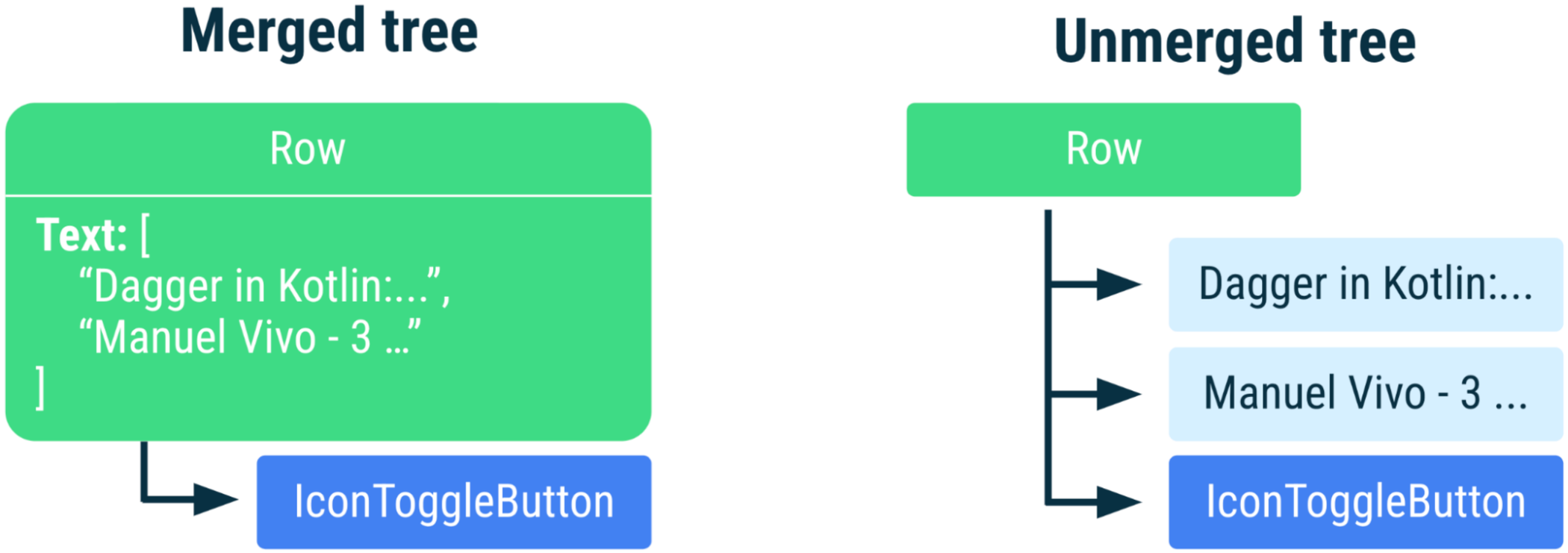
Какие видим проблемы в существующем коде

1. Нужно везде пробрасывать ComposeTestRule

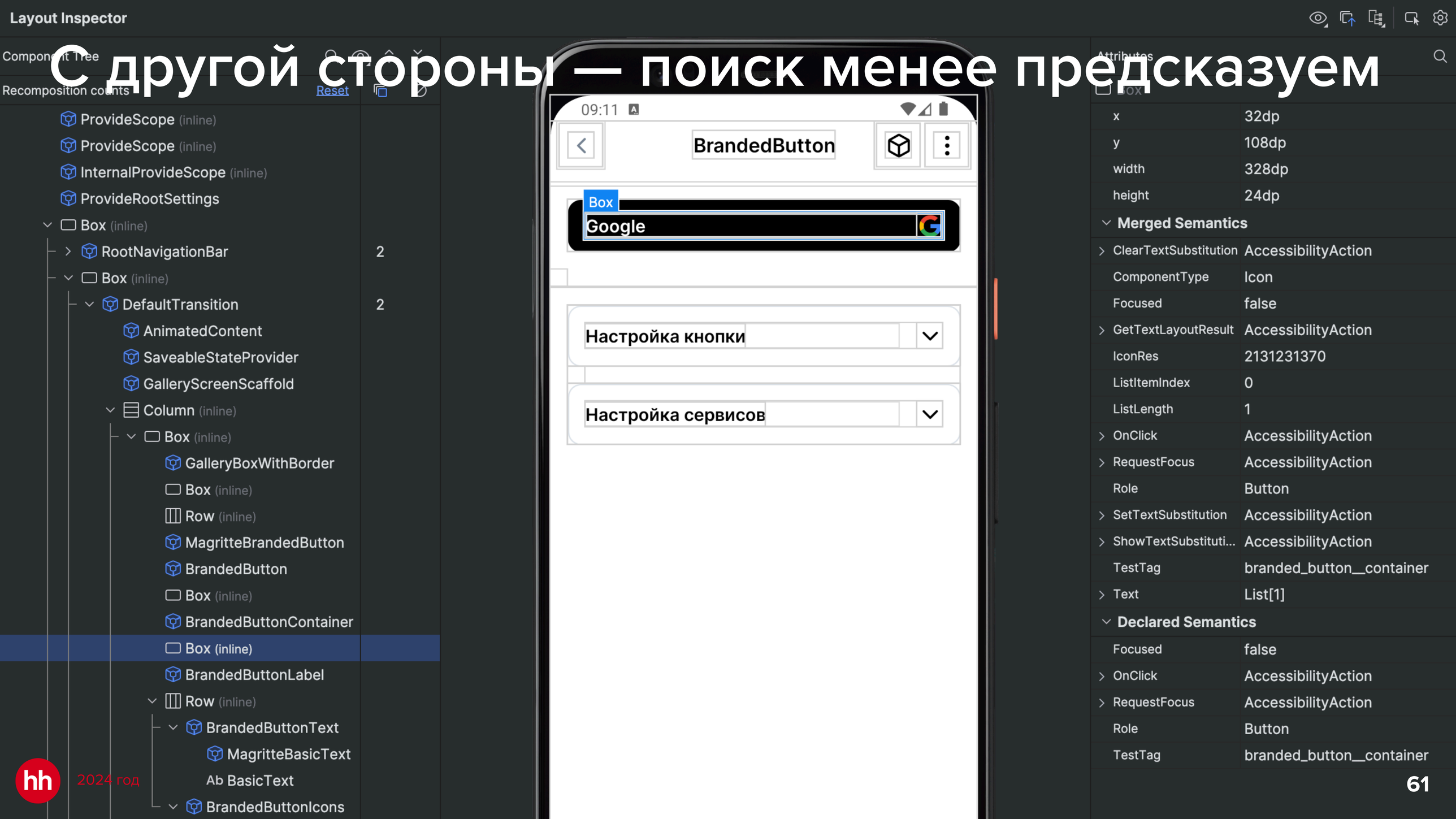
2. **По умолчанию используется merged-дерево семантики**

→ Ввели контейнер для хранения ComposeTestRule между тестами

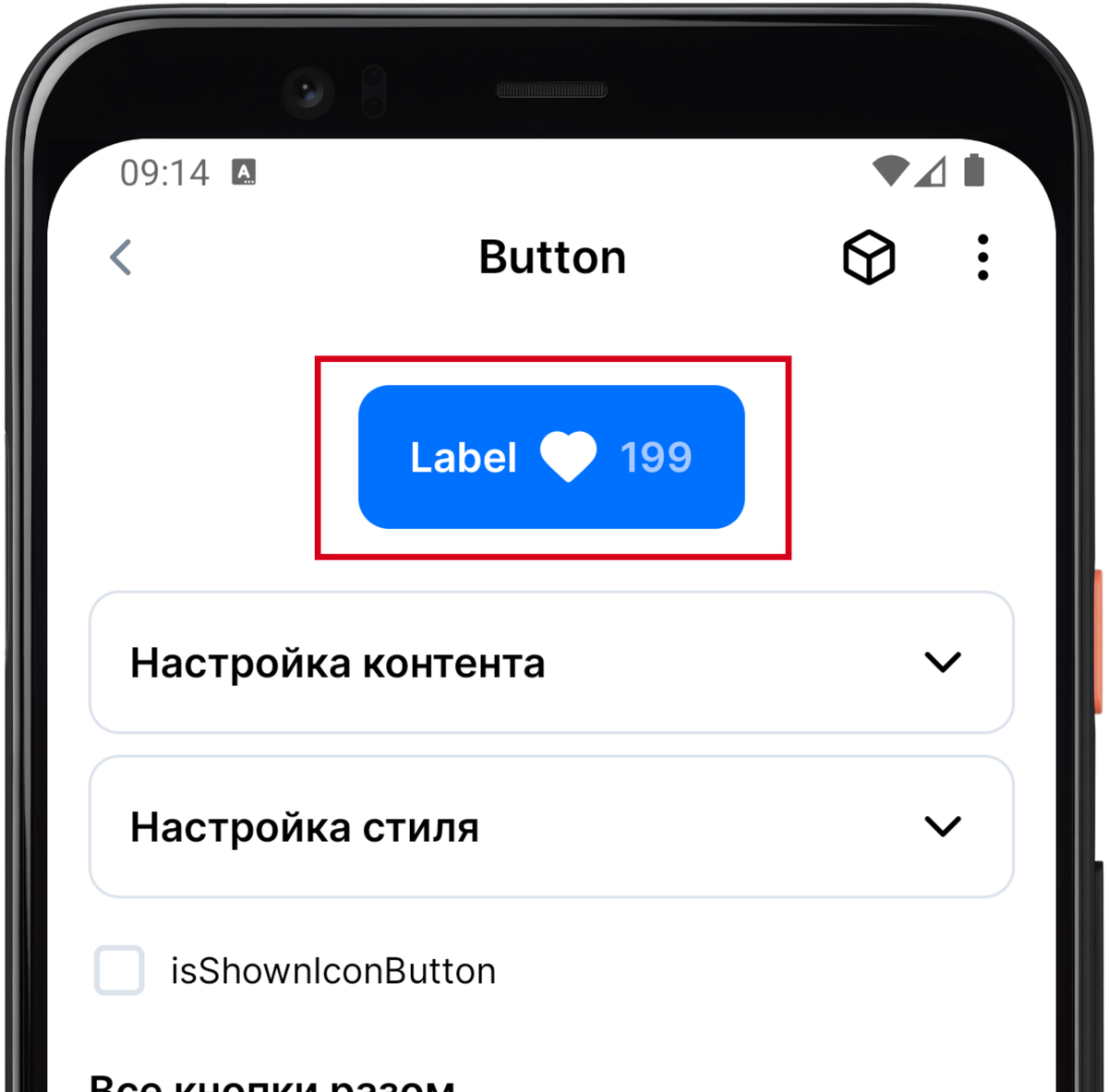
С одной стороны — merged-дерево это отлично



С другой стороны — поиск менее предсказуем

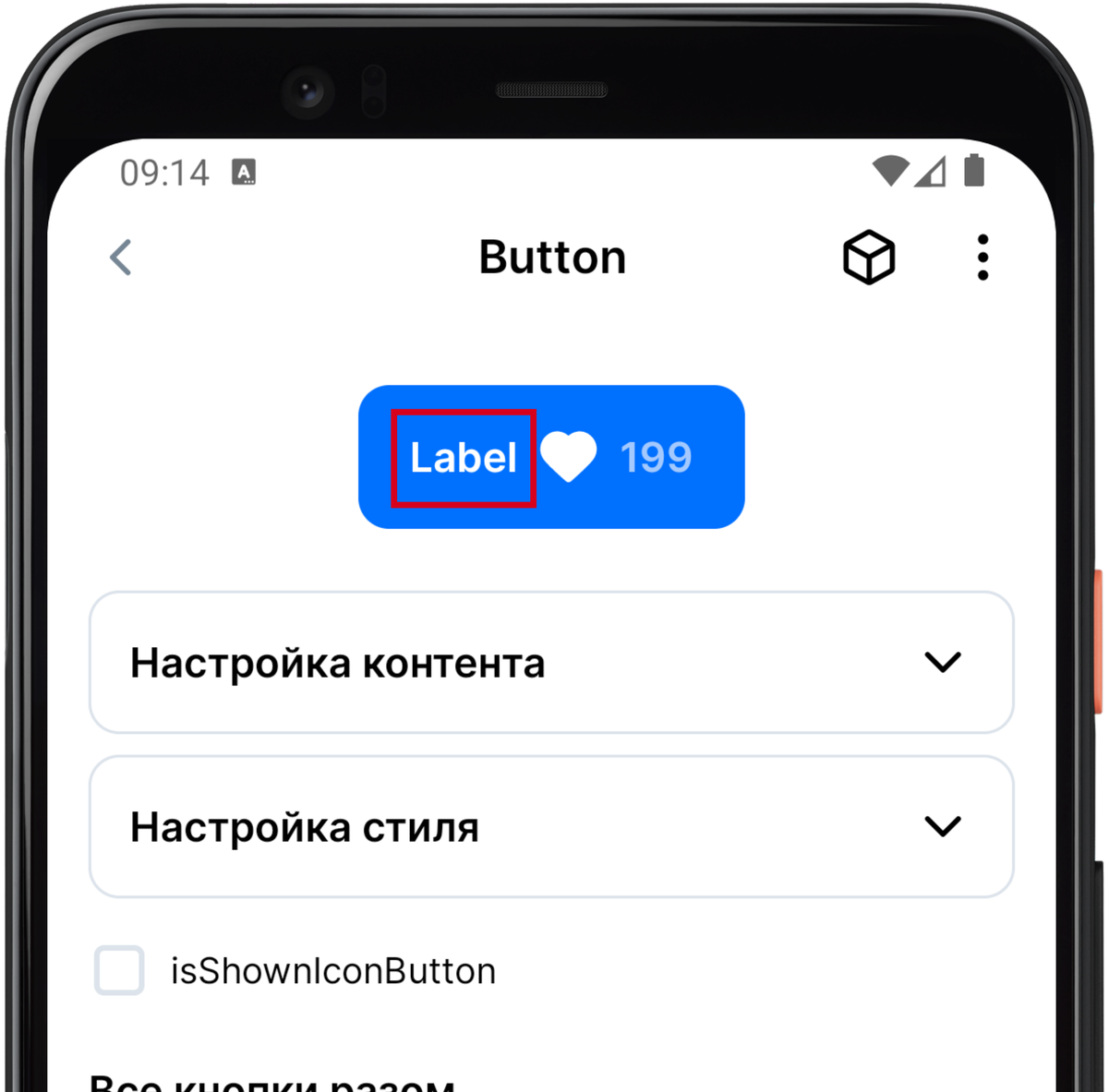


С другой стороны — поиск менее предсказуем



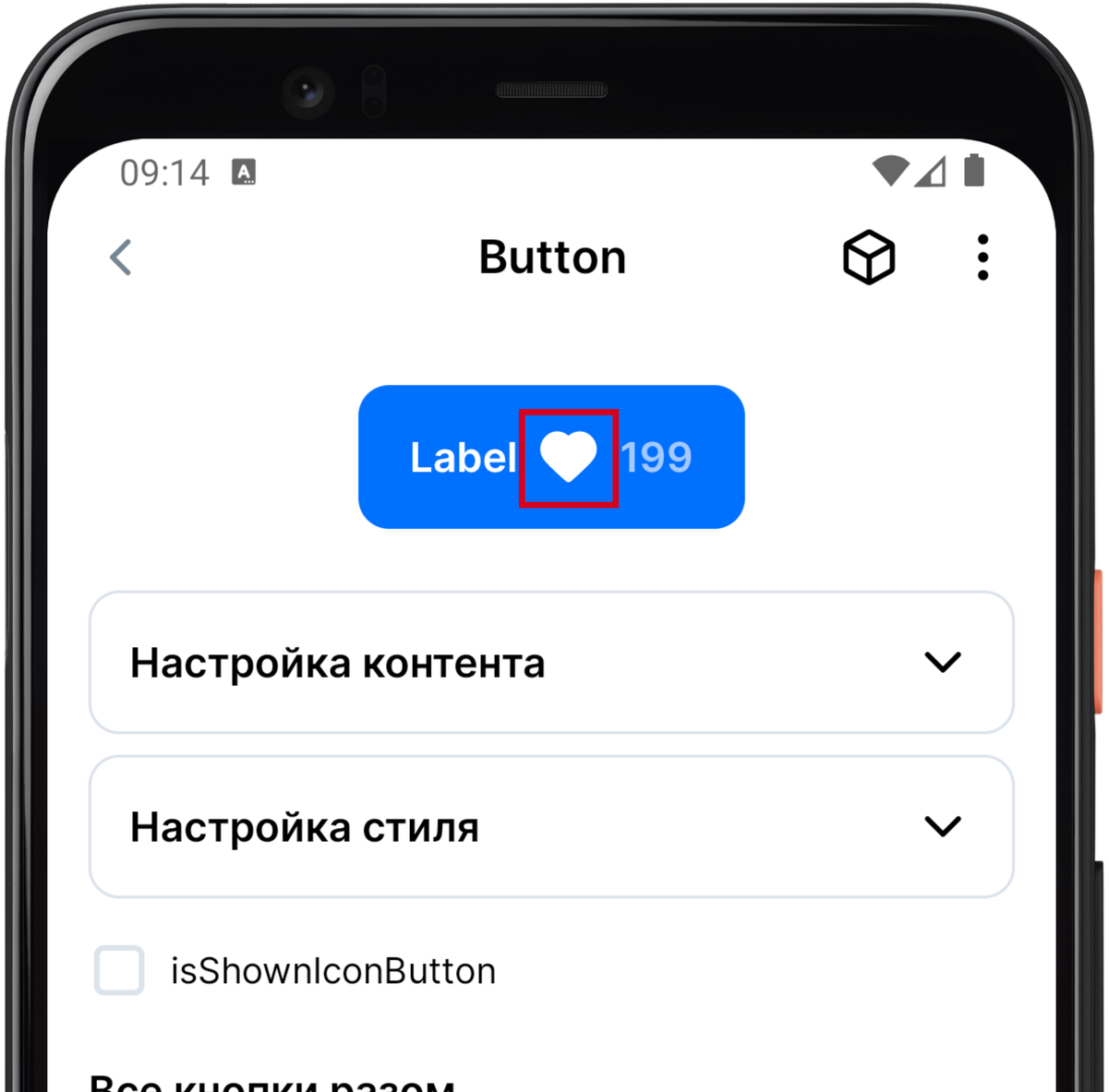
Представьте себе обычную кнопку

1 Есть надпись



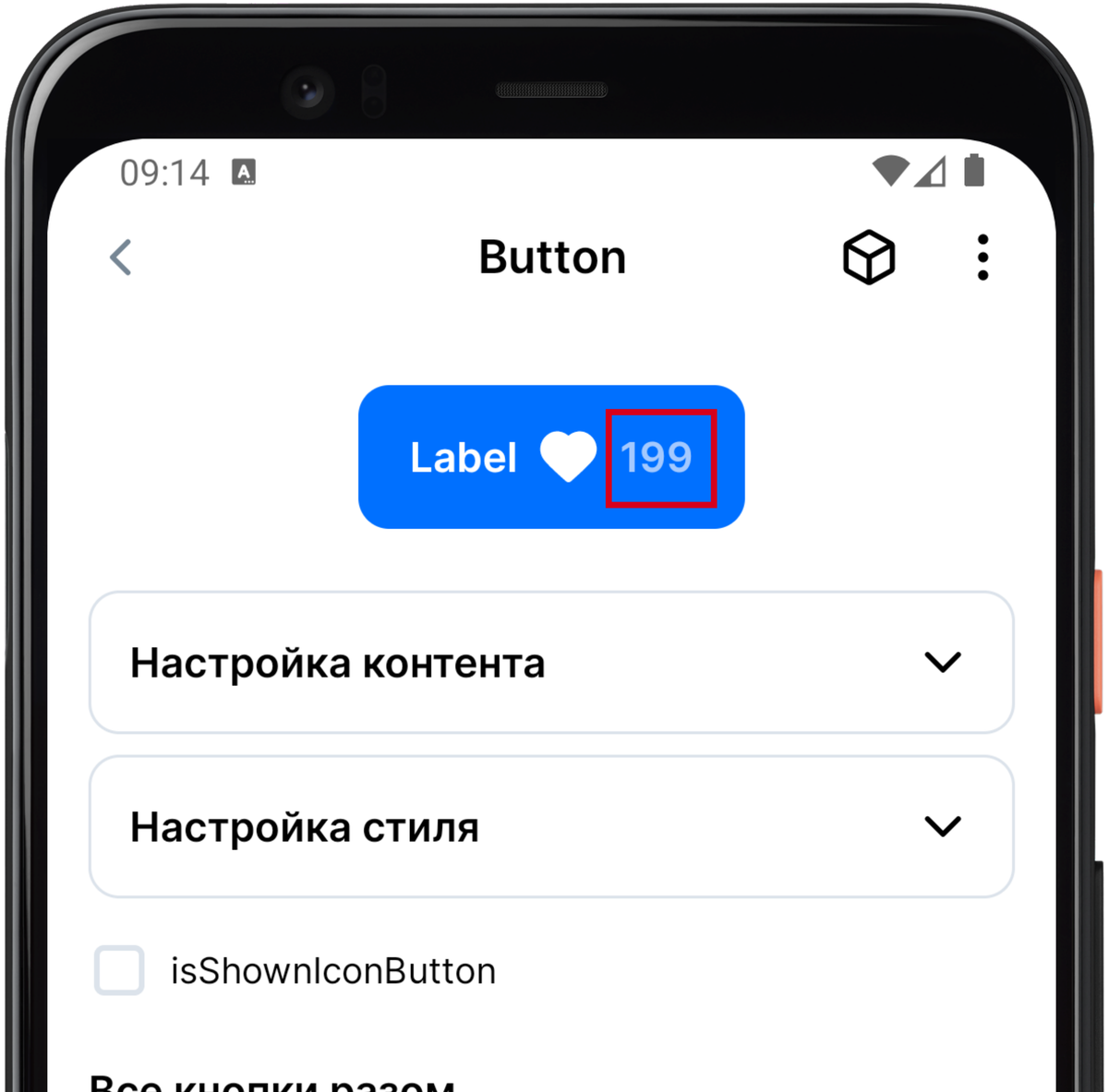
Представьте себе обычную кнопку

1. Есть надпись
2. Есть иконка



Представьте себе обычную кнопку

1. Есть надпись
2. Есть иконка
3. Есть текст счётчика



Это логично — описать поиск счётчика вот так

```
inner class Checks : PageObjectIntentions<Checks>() {

    fun TestContext<*>.checkForError() {
        step("Ошибка вышла, вот о чём молчит наука!") {
            // Нашли саму кнопку
            val buttonNode = child<KNode> {
                hasTestTag(GalleryMagritteTestTags.button.mainButton)
            }
            // Ищем надпись внутри кнопки
            val buttonPostfix = buttonNode.child<KNode> {
                hasTestTag(MagritteTestTags.button.postfix)
            }
            // Проверяем текст надписи
            buttonPostfix.assertTextEquals(«199")
        }
    }
```

Но мы получаем краш

```
----- begin exception -----
        java.lang.AssertionError: Failed to assert the following: (Text + EditableText = [199])
        Can't retrieve node at index '0' of '((hasAnyAncestorThat(TestTag =
'button_screen__button__main')) && (hasAnyAncestorThat(Empty matcher))) && (TestTag =
'button__postfix')'
        There are no existing nodes for that selector.
```

Как выглядит дерево семантики?

```
val buttonNode = child<KNode> {
    hasTestTag(GalleryMagritteTestTags.button.mainButton)
}
buttonNode.printToLog(tag = "TEST")
```

Как выглядит дерево семантики?

```
val buttonNode = child<KNode> {
    hasTestTag(GalleryMagritteTestTags.button.mainButton)
}
buttonNode.printToLog(tag = "TEST")
----------------------------
printToLog:
    Printing with useUnmergedTree = 'false'
    Node #276 at (l=333.0, t=264.0, r=747.0, b=418.0)px,
        Tag: 'button_screen__button__main'
    ContentDescription = '[199 лайков]'
    ComponentType = 'Button'
    Role = 'Button'
    Focused = 'false'
    Actions = [OnClick, RequestFocus]
    MergeDescendants = 'true'
```

А где всё?..

```
val buttonNode = child<KNode> {
    hasTestTag(GalleryMagritteTestTags.button.mainButton)
}
buttonNode.printToLog(tag = "TEST")
----------------------------
printToLog:
    Printing with useUnmergedTree = 'false'
    Node #276 at (l=333.0, t=264.0, r=747.0, b=418.0)px,
        Tag: 'button_screen__button__main'
    ContentDescription = '[199 лайков]'
    ComponentType = 'Button'
    Role = 'Button'
    Focused = 'false'
    Actions = [OnClick, RequestFocus]
    MergeDescendants = 'true'
```

Но стоит мне использовать useUnmergedTree...

```
inner class Checks : PageObjectIntentions<Checks>() {

    fun TestContext<*>.checkForError() {
        step("Ошибка вышла, вот о чём молчит наука!") {
            // Нашли саму кнопку
            val buttonNode = child<KNode> {
                hasTestTag(GalleryMagritteTestTags.button.mainButton)
                useUnmergedTree = true // Работаю с несмёрдженным деревом
            }
            // Ищем надпись внутри кнопки
            val buttonPostfix = buttonNode.child<KNode> {
                hasTestTag(MagritteTestTags.button.postfix)
            }
            // Проверяем текст надписи
            buttonPostfix.assertTextEquals("199")
        }
    }
```

И дерево начинает выглядеть предсказуемо

```
I TEST STEP: "3.1. Ошибка вышла, вот о чём молчит наука!" in GalleryButtonScreenTest
D printToLog:
  Printing with useUnmergedTree = 'true'
  Node #423 at (l=333.0, t=264.0, r=747.0, b=418.0)px, Tag: 'button_screen__button__main'
  ContentDescription = '[199 лайков]'
  ComponentType = 'Button'
  Role = 'Button'
  Focused = 'false'
  Actions = [OnClick, RequestFocus]
  MergeDescendants = 'true'
   |-Node #424 at (l=388.0, t=308.0, r=692.0, b=374.0)px, Tag: 'button__content'
     ClearAndSetSemantics = 'true'
      |-Node #425 at (l=388.0, t=311.0, r=503.0, b=372.0)px
      | MergeDescendants = 'true'
      |  |-Node #426 at (l=388.0, t=311.0, r=503.0, b=372.0)px, Tag: 'button__label'
      |    Text = '[Label]'
      |    Actions = [SetTextSubstitution, ShowTextSubstitution, ClearTextSubstitution, GetTextLayoutResult]
      |-Node #427 at (l=525.0, t=308.0, r=591.0, b=374.0)px, Tag: 'button__icon'
      | IconRes = '2131231384'
      | IconTint = 'Color(1.0, 1.0, 1.0, 1.0, sRGB IEC61966-2.1)'
      | ComponentType = 'Icon'
      |-Node #428 at (l=613.0, t=311.0, r=692.0, b=372.0)px, Tag: 'button__postfix'
        Text = '[199]'
        Actions = [SetTextSubstitution, ShowTextSubstitution, ClearTextSubstitution, GetTextLayoutResult]
```

Вроде всё логично, но как-то больно

```
private val contentSettings = KNode {
    hasText("Настройка контента")
    hasClickAction()
    useUnmergedTree = true
}
```

```
private val clearIcon = KNode {
    hasTestTag("clearIcon")
    useUnmergedTree = true
}
```

В итоге решили всегда работать с unmerged tree

В будущих версиях Kakao...

```
KakaoCompose.Override.useUnmergedTree = true
```

Какие видим проблемы в существующем коде

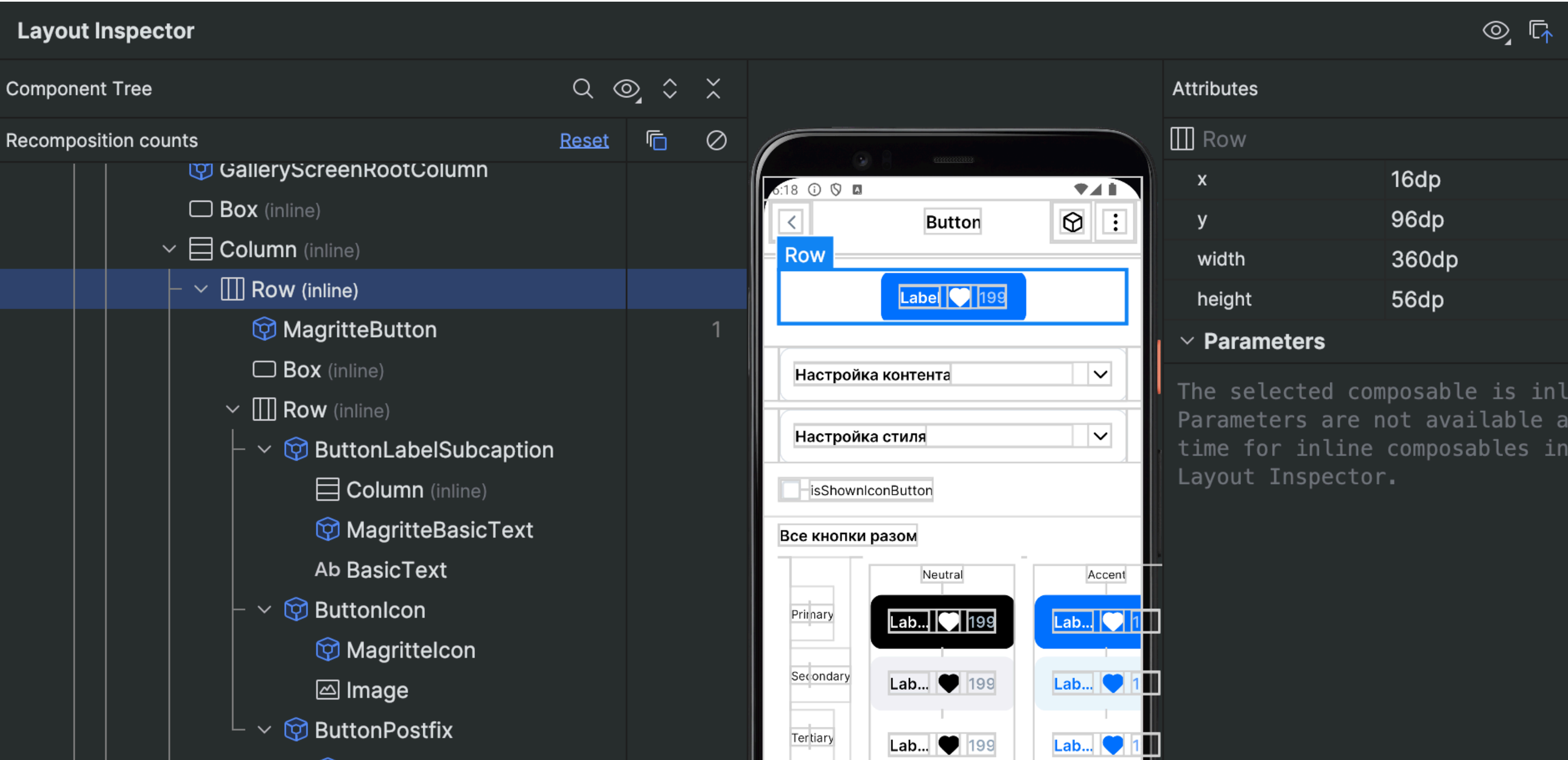
1. Нужно везде пробрасывать ComposeTestRule → Ввели контейнер для хранения ComposeTestRule между тестами

2. **По умолчанию используется merged-дерево семантики** → **Всегда работаем с unmerged-деревом семантики**

Какие видим проблемы в существующем коде

1. Нужно везде пробрасывать ComposeTestRule → Ввели контейнер для хранения ComposeTestRule между тестами
2. По умолчанию используется merged-дерево семантики → Всегда работаем с unmerged-деревом семантики
3. **Непонятно, как работать с компонентами дизайн-системы**

Каждый компонент имеет свою структуру



QA просят схожий с Page Object API

Отдельные KNode для каждого компонента

```
class KStandaloneLinkNode(
    semanticsProvider: SemanticsNodeInteractionsProvider,
    nodeMatcher: NodeMatcher,
    parentNode: BaseNode<*>?,
) : BaseNode<KStandaloneLinkNode>(semanticsProvider, nodeMatcher, parentNode) {

    val actions = Actions()
    val checks = Checks()

    // region Внутренние элементы
    // ...
    // endregion

    inner class Actions : PageObjectIntentions<Actions>() {
        // ...
    }

    inner class Checks : PageObjectIntentions<Checks>() {
        // ...
    }
```

Стандартное API создания KNode неудобно

```
// region Внутренние элементы

private val mainButton = child<KButtonNode> {
    hasTestTag(GalleryMagritteTestTags.button.mainButton)
}

private val mainButton2 : KButtonNode = child {
    hasTestTag(GalleryMagritteTestTags.button.mainButton)
}

// endregion
```

Сделали удобные factory-методы

```
// region Дополнительные factory-методы

fun BaseNode<*>.KButtonNode(
    viewBuilderAction: ViewBuilder.() → Unit
): KButtonNode = child {
    hasComponentType(SemanticsComponentType.Button)
    viewBuilderAction.invoke(this)
}

fun BaseNode<*>.KButtonNode(testTag: String): KButtonNode =
    KButtonNode { hasTestTag(testTag) }

// endregion
```

Используем KNode в Page Object

```
private val mainButton = KButtonNode(
    GalleryMagritteTestTags.button.mainButton
)

inner class Checks : PageObjectIntentions<Checks>() {

    fun TestContext<*>.checkButtonLabel(label: String) {
        step("Проверяем текст надписи на итоговой кнопке (label: '$label')") {
            mainButton.checks {
                checkLabel(text = label)
            }
        }
    }
}
```

Удобно создать KNode

```
private val mainButton = KButtonNode(
    GalleryMagritteTestTags.button.mainButton
)

inner class Checks : PageObjectIntentions<Checks>() {

    fun TestContext<*>.checkButtonLabel(label: String) {
        step("Проверяем текст надписи на итоговой кнопке (label: '$label')") {
            mainButton.checks {
                checkLabel(text = label)
            }
        }
    }
}
```

Удобно используем

```
private val mainButton = KButtonNode(
    GalleryMagritteTestTags.button.mainButton
)

inner class Checks : PageObjectIntentions<Checks>() {

    fun TestContext<*>.checkButtonLabel(label: String) {
        step("Проверяем текст надписи на итоговой кнопке (label: '$label')") {
            mainButton.checks {
                checkLabel(text = label)
            }
        }
    }
}
```

Какие видим проблемы в существующем коде

1	Нужно везде пробрасывать ComposeTestRule	→ Ввели контейнер для хранения ComposeTestRule между тестами
2	По умолчанию используется merged-дерево семантики	→ Всегда работаем с unmerged-деревом семантики
3	**Непонятно как работать с компонентами дизайн-системы**	→ **Для каждого компонента пишем отдельные KNode (мини Page Object)**

Какие видим проблемы в существующем коде

1	Нужно везде пробрасывать ComposeTestRule	→ Ввели контейнер для хранения ComposeTestRule между тестами
2	По умолчанию используется merged-дерево семантики	→ Всегда работаем с unmerged-деревом семантики
3	Непонятно как работать с компонентами дизайн-системы	→ Для каждого компонента пишем отдельные KNode (мини Page Object)
4	**Неудобное API для работы с Lazy-списками в Kakao**	

Как выглядит объявление узла Lazy-списка

```
private val list = KLazyListNode(
    semanticsProvider = composeTestRule,
    viewBuilderAction = {
        hasTestTag(MarketplaceMentorListTestTags.list.lazyList)
    },
    positionMatcher = { position → hasTestTag("position=$position") }
    itemTypeBuilder = {
        itemType(MarketplaceMentorsListComposePageObject::LazyListItemNode)
    },
)
```

Нужно пробросить ComposeTestRule

```
private val list = KLazyListNode(
    semanticsProvider = composeTestRule,
    viewBuilderAction = {
        hasTestTag(MarketplaceMentorListTestTags.list.lazyList)
    },
    positionMatcher = { position -> hasTestTag("position=$position") },
    itemTypeBuilder = {
        itemType(MarketplaceMentorsListComposePageObject::LazyListItemNode)
    },
)
```

Поиск узла с Lazy-списком

```
private val list = KLazyListNode(
    semanticsProvider = composeTestRule,
    viewBuilderAction = {
        hasTestTag(MarketplaceMentorListTestTags.list.lazyList)
    },
    positionMatcher = { position -> hasTestTag("position=$position") },
    itemTypeBuilder = {
        itemType(MarketplaceMentorsListComposePageObject::LazyListItemNode)
    },
)
```

Указание Matcher для поиска элемента списка

```
private val list = KLazyListNode(
    semanticsProvider = composeTestRule,
    viewBuilderAction = {
        hasTestTag(MarketplaceMentorListTestTags.list.lazyList)
    },
    positionMatcher = { position -> hasTestTag("position=$position") },
    itemTypeBuilder = {
        itemType(MarketplaceMentorsListComposePageObject::LazyListItemNode)
    },
)
```

Перечисление типов элементов списка

```
private val list = KLazyListNode(
    semanticsProvider = composeTestRule,
    viewBuilderAction = {
        hasTestTag(MarketplaceMentorListTestTags.list.lazyList)
    },
    positionMatcher = { position -> hasTestTag("position=$position") },
    itemTypeBuilder = {
        itemType(MarketplaceMentorsListComposePageObject::LazyListItemNode)
    },
)
```

Как выглядит описание элемента списка

```
private class LazyListItemNode(
    semanticsNode: SemanticsNode,
    semanticsProvider: SemanticsNodeInteractionsProvider,
) : KLazyListItemNode<LazyListItemNode>(semanticsNode, semanticsProvider) {

    val title = KNode {
        hasTestTag("title")
    }

    val subtitleButton = KButtonNode {
        hasTestTag("subtitleButton")
    }

}
```

Конструктор элемента

```
private class LazyListItemNode(
    semanticsNode: SemanticsNode,
    semanticsProvider: SemanticsNodeInteractionsProvider,
) : KLazyListItemNode<LazyListItemNode>(semanticsNode, semanticsProvider) {

    val title = KNode {
        hasTestTag("title")
    }

    val subtitleButton = KButtonNode {
        hasTestTag("subtitleButton")
    }

}
```

Описание внутренних элементов

```
private class LazyListItemNode(
    semanticsNode: SemanticsNode,
    semanticsProvider: SemanticsNodeInteractionsProvider,
) : KLazyListItemNode<LazyListItemNode>(semanticsNode, semanticsProvider) {

    val title = KNode {
        hasTestTag("title")
    }

    val subtitleButton = KButtonNode {
        hasTestTag("subtitleButton")
    }

}
```

Использование API KLazyListNode

```
private val list = KLazyListNode(/* ... */)

@OptIn(ExperimentalTestApi::class)
fun TestContext<*>.checkCardExistsByPosition(position: Int) {
    step("Проверяем отображение карточки с номером $position") {
        list {
            childAt<LazyListItemNode>(position) {
                assertIsDisplayed()
                assertHasClickAction()
            }

            val item = childWith<LazyListItemNode> {
                hasAnyDescendant(SemanticsMatcher.hasText("Some text"))
            }
            item {
                assertIsDisplayed()
            }
        }
    }
}
```

Поиск элемента по индексу

```
private val list = KLazyListNode(/* ... */)

@OptIn(ExperimentalTestApi::class)
fun TestContext<*>.checkCardExistsByPosition(position: Int) {
    step("Проверяем отображение карточки с номером $position") {
        list {
            childAt<LazyListItemNode>(position) {
                assertIsDisplayed()
                assertHasClickAction()
            }

            val item = childWith<LazyListItemNode> {
                hasAnyDescendant(SemanticsMatcher.hasText("Some text"))
            }
            item {
                assertIsDisplayed()
            }
        }
    }
}
```

Поиск элемента по Matcher

```
private val list = KLazyListNode(/* ... */)

@OptIn(ExperimentalTestApi::class)
fun TestContext<*>.checkCardExistsByPosition(position: Int) {
    step("Проверяем отображение карточки с номером $position") {
        list {
            childAt<LazyListItemNode>(position) {
                assertIsDisplayed()
                assertHasClickAction()
            }

            val item = childWith<LazyListItemNode> {
                hasAnyDescendant(SemanticsMatcher.hasText("Some text"))
            }
            item {
                assertIsDisplayed()
            }
        }
    }
}
```

Что нам не нравится?

1. Всегда требуется указывать типы элементов списка
2. Неконсистентное API работы с элементами
3. Конструктор элемента списка вербозен

Пишем немножко кода...

```
fun BaseNode<*>.KLazyListNode(
    semanticsProvider: SemanticsNodeInteractionsProvider = ComposeRuleContainer.getComposeRule(),
    positionMatcher: (position: Int) -> SemanticsMatcher = { position ->
        hasTestTag("position=$position")
    },
    itemTypeBuilder: KLazyListItemBuilder.() -> Unit = {},
    viewBuilderAction: ViewBuilder.() -> Unit,
): KLazyListNode {
    return KLazyListNode(
        semanticsProvider = semanticsProvider,
        viewBuilderAction = viewBuilderAction,
        positionMatcher = positionMatcher,
        itemTypeBuilder = {
            itemType(::CommonLazyListItemNode)
            itemTypeBuilder.invoke(this)
        }
    )
}
```

```
@OptIn(ExperimentalTestApi::class)
inline fun <reified T> KLazyListNode.childWith(
    noinline childMatcher: ViewBuilder.() -> Unit,
    action: (T) -> Unit,
) where T :KLazyListItemNode<*> {
    val item = childWith<T>(childMatcher)
    action.invoke(item)
}
```

```
class CommonLazyListItemNode(
    semanticsNode: SemanticsNode,
    semanticsProvider: SemanticsNodeInteractionsProvider,
) : KLazyListItemNode<CommonLazyListItemNode>(
    semanticsNode, semanticsProvider
)
```

И API стало выглядеть приятнее

```
private val list = KLazyListNode {
    hasTestTag("my_list")
}

fun TestContext<*>.clickOnItemWithTitle(title: String) {
    step("Работа с элементом списка") {
        list {
            childWith<CommonLazyListItemNode>(
                { hasAnyDescendant(SemanticsMatcher.hasText(title)) }
            ) {
                performClick()
            }
        }
    }
}
```

Не все боли получилось решить, но это ок

1. Всегда требуется указывать типы элементов списка
2. Неконсистентное API работы с элементами
3. Конструктор элемента списка вербозен

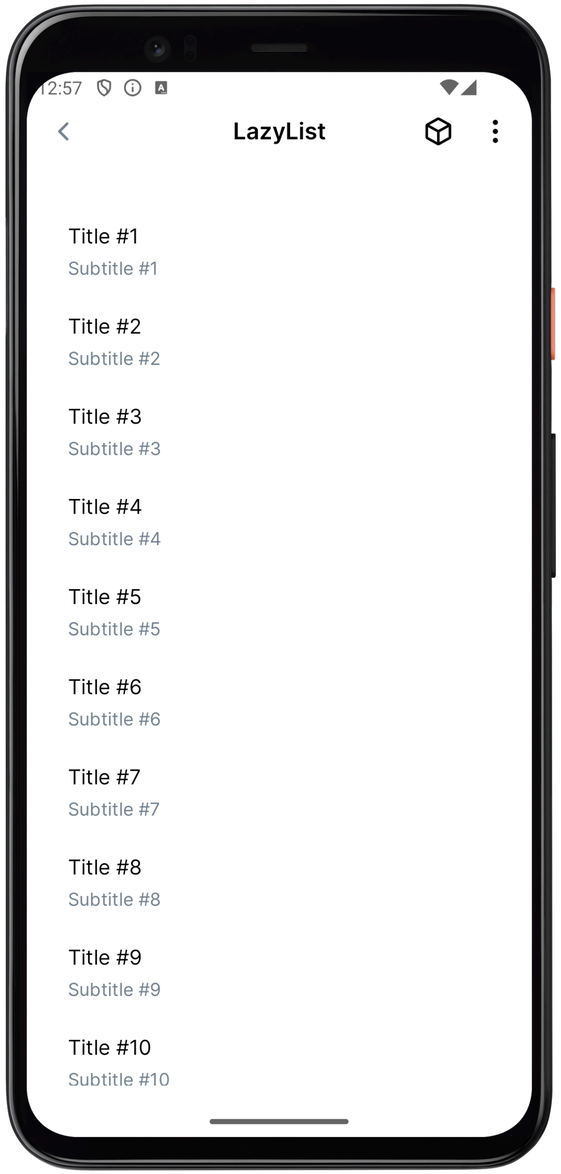
Какие видим проблемы в существующем коде

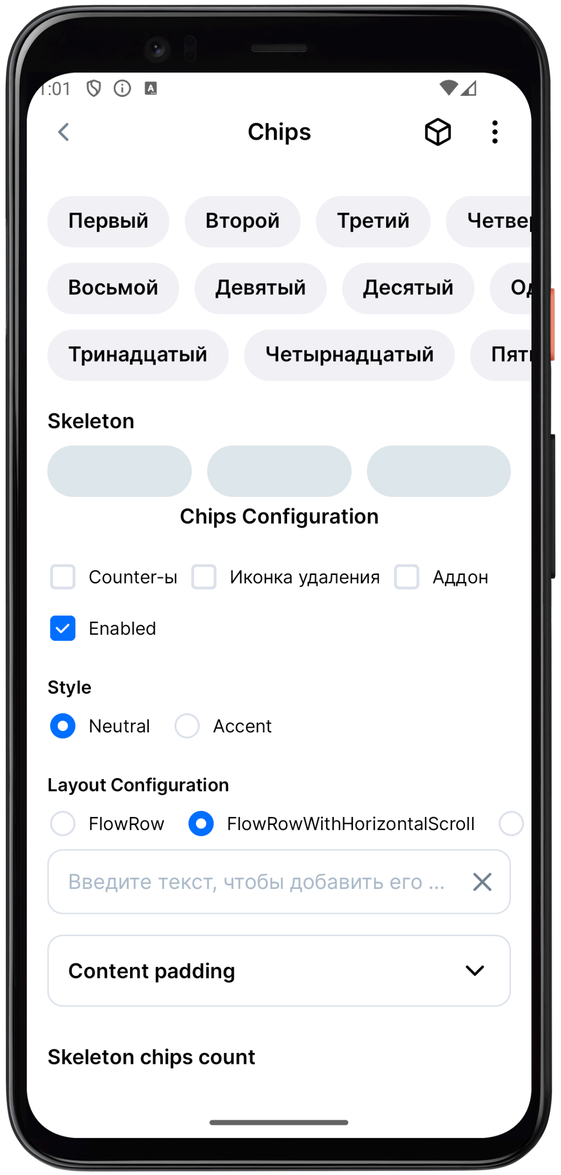
1	Нужно везде пробрасывать ComposeTestRule	→ Ввели контейнер для хранения ComposeTestRule между тестами
2	По умолчанию используется merged-дерево семантики	→ Всегда работаем с unmerged-деревом семантики
3	Непонятно как работать с компонентами дизайн-системы	→ Для каждого компонента пишем отдельные KNode (мини Page Object)
4	**Неудобное API для работы с Lazy-списками в Kakao**	→ **Подправили использование API, дописав немного функций и классов**

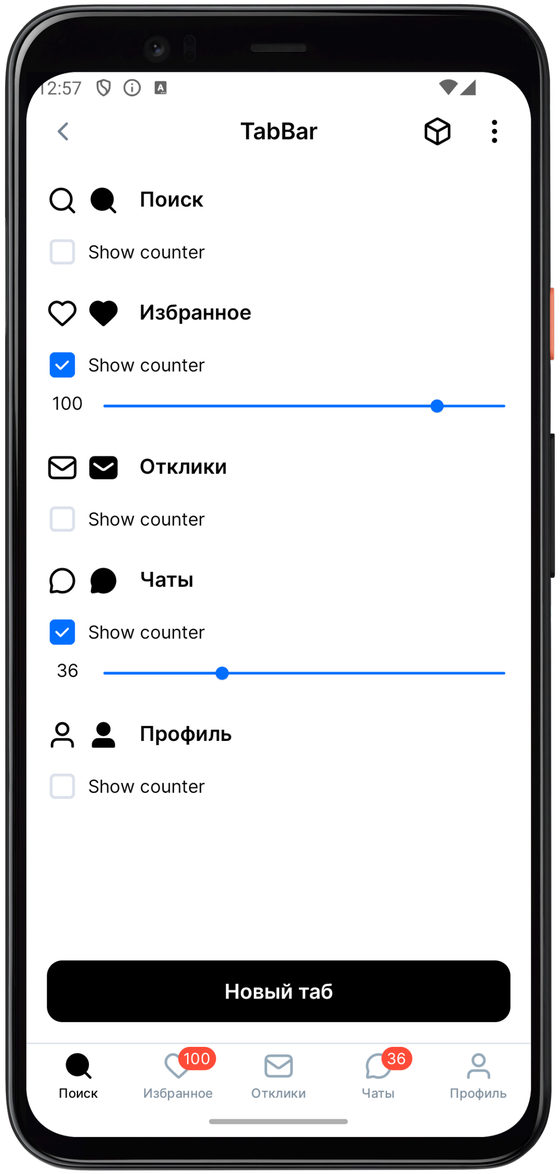
Какие видим проблемы в существующем коде

1	Нужно везде пробрасывать ComposeTestRule	→ Ввели контейнер для хранения ComposeTestRule между тестами
2	По умолчанию используется merged-дерево семантики	→ Всегда работаем с unmerged-деревом семантики
3	Непонятно как работать с компонентами дизайн-системы	→ Для каждого компонента пишем отдельные KNode (мини Page Object)
4	Неудобное API для работы с Lazy-списками в Kakao	→ Подправили использование API, дописав немного функций и классов
5	**Неудобное API для работы с обычными списками в Kakao**	

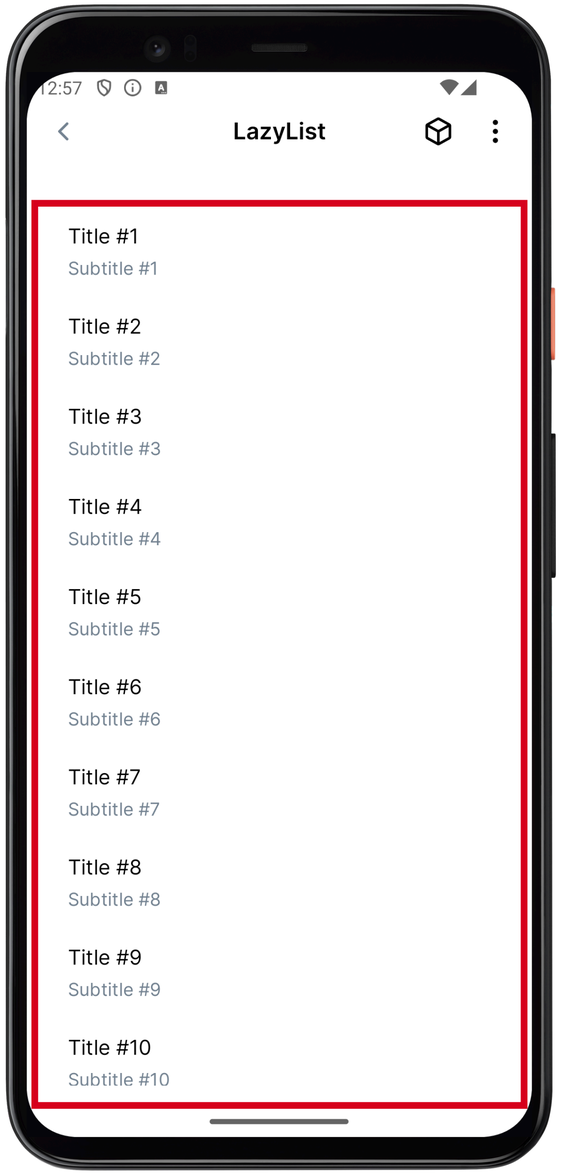
Списки в приложении бывают разные

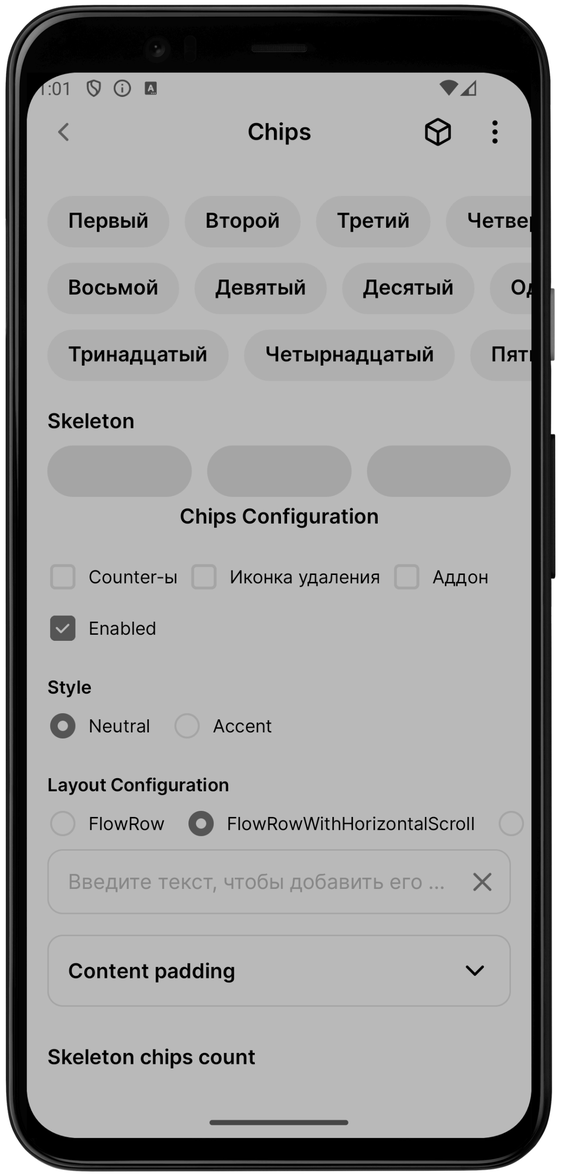


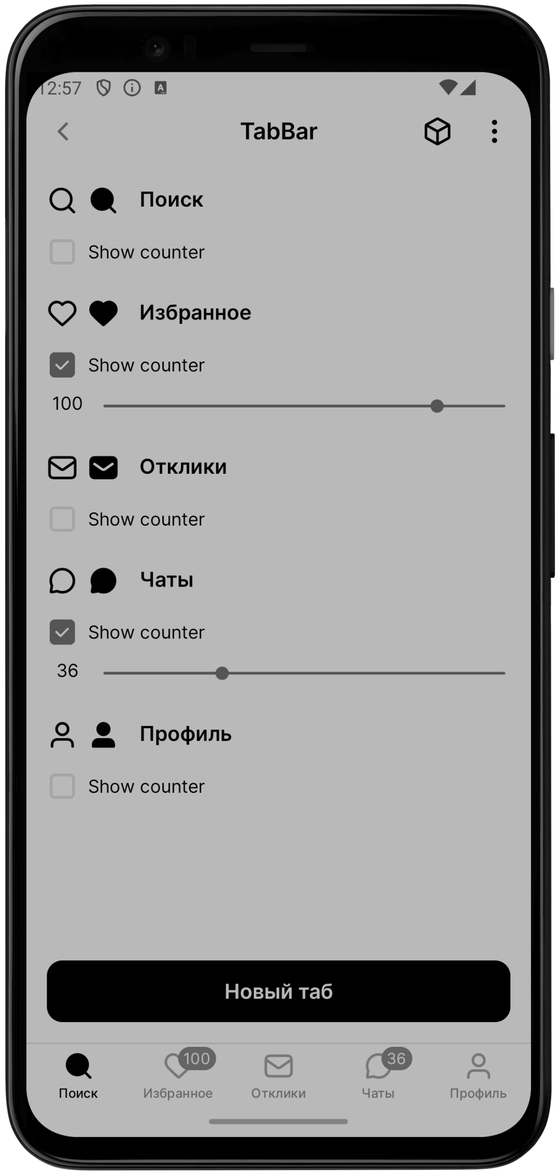




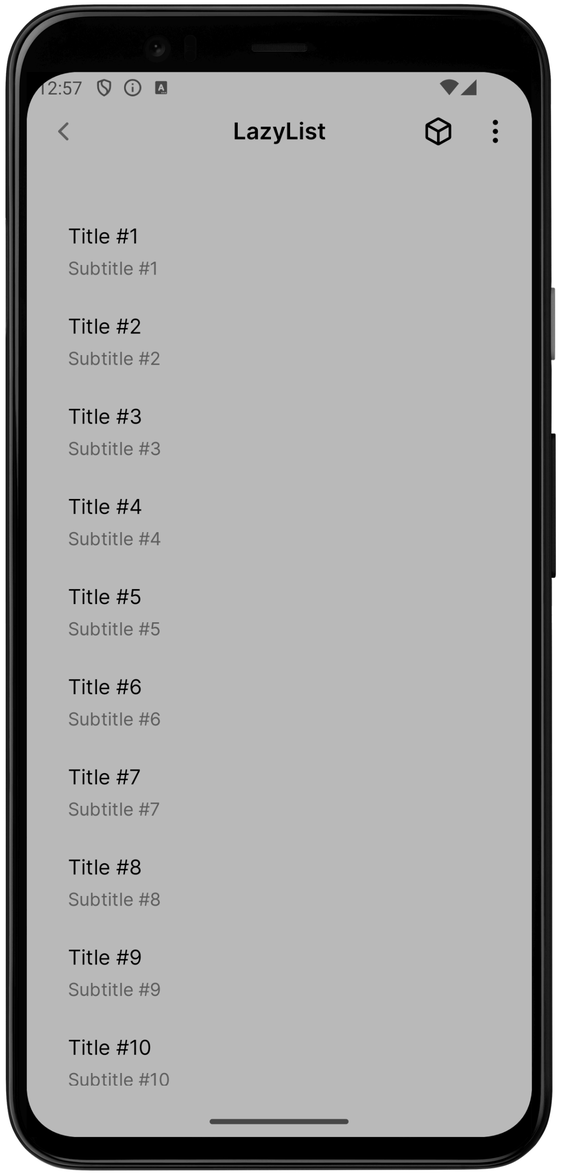
Списки бывают lazy

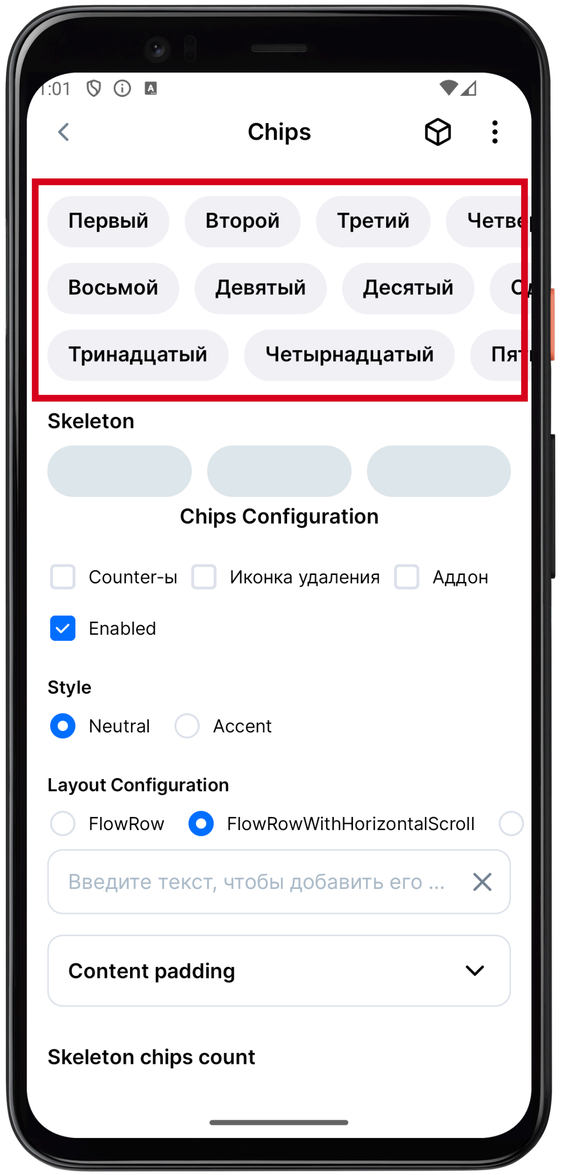


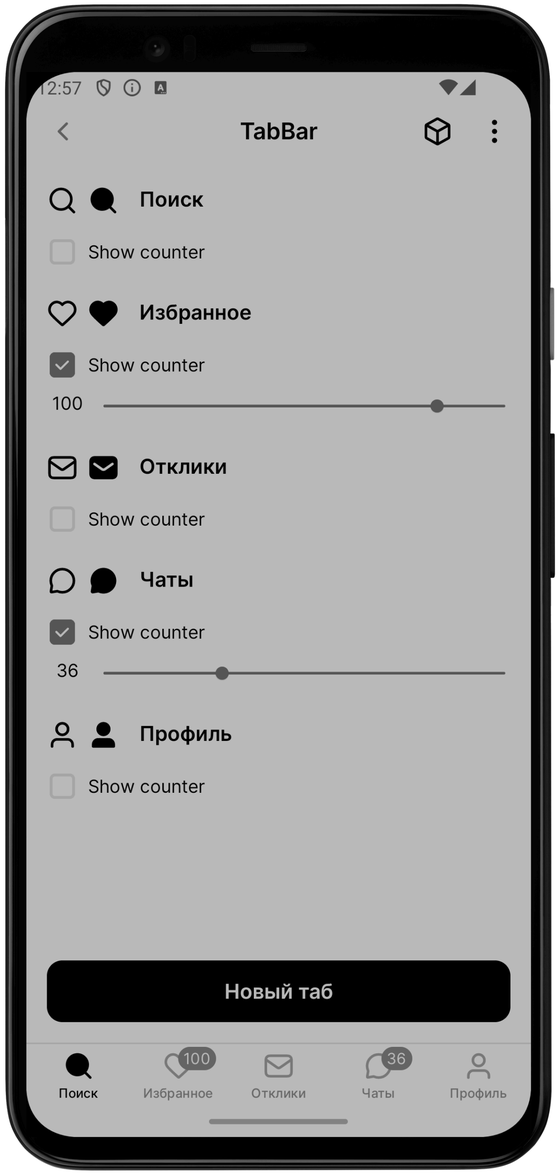




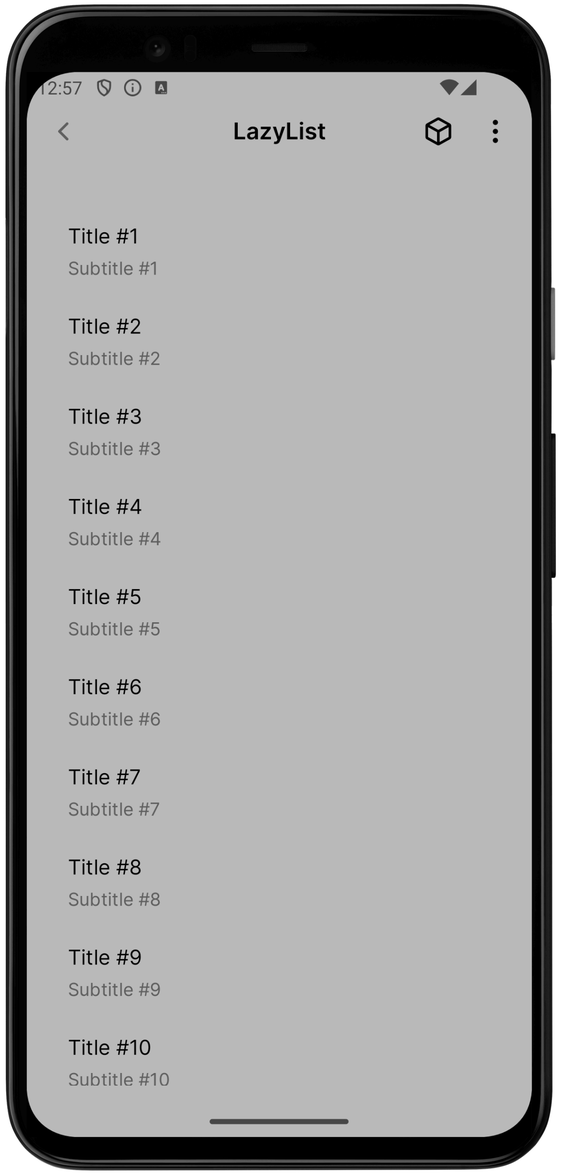
Списки бывают скроллящимися

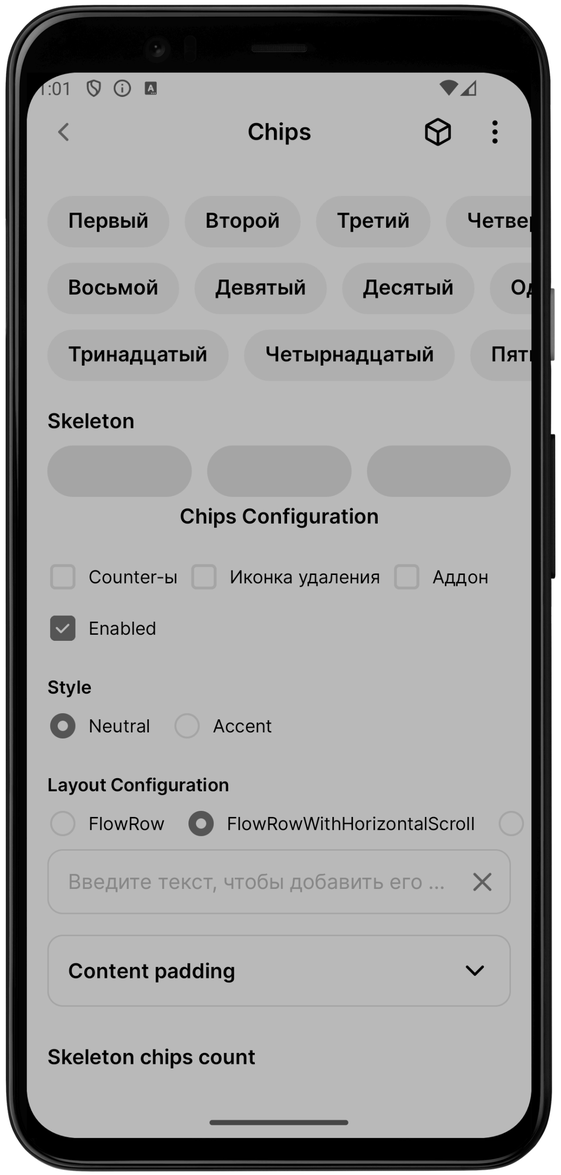


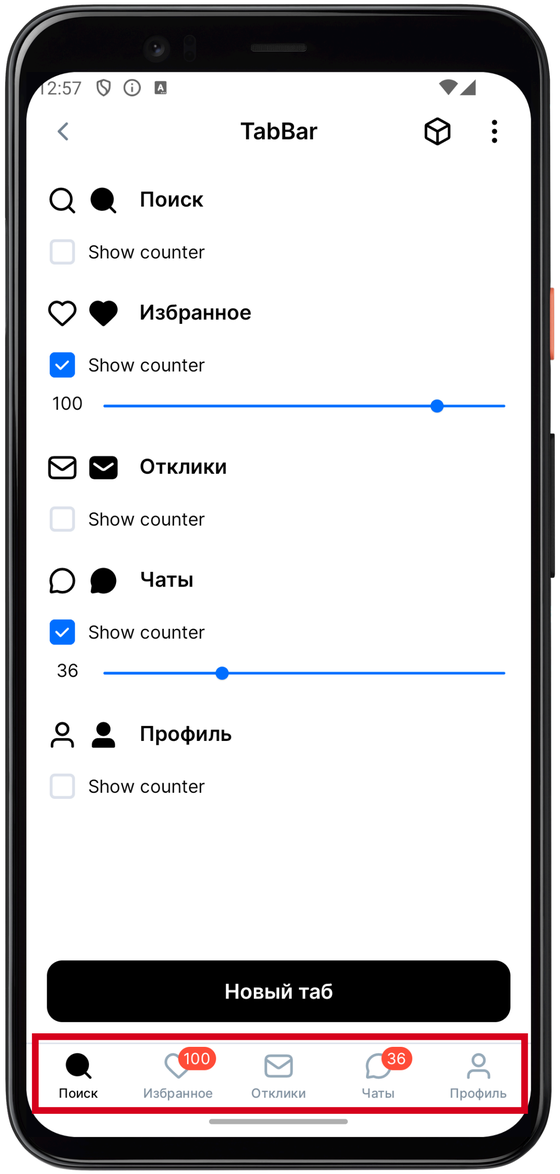




А ещё они бывают НЕскроллящимися







KLazyListNode не подходит для not scrollable

```
@ExperimentalTestApi
inline fun <reified T : KLazyListItemNode<*>> childWith(noinline childMatcher
ViewBuilder.() → Unit): T {
    val provideItem = itemTypes.getOrElse(T::class) {
        throw LazyListItemProvisionException(T::class)
    }.provideItem

    val nodeMatcher = ViewBuilder().apply(childMatcher).build()

    // Всегда пытаемся выполнить подскролл к элементу
    // с помощью SemanticsActions.ScrollBy
    performScrollToNode(nodeMatcher.matcher)

    val semanticsNode = semanticsProvider
        .onNode(semanticsMatcher)
        .onChildren()
        .filter(nodeMatcher.matcher)[nodeMatcher.position]
```

Придётся копировать исходники к себе

Может попробуем исправить все проблемы?

1. Всегда требуется указывать типы элементов списка
2. Неконсистентное API работы с элементами
3. Конструктор элемента списка вербозен
4. KLazyListNode не подходит для not scrollable списков

Новое API для работы со списками — KListNode

```
private val screenList = KListNode(isScrollable = false) {
    hasTestTag(MarketplaceMentorListTestTags.list.lazyList)
}

// или
private val screenList = KListNode(
    testTag = MarketplaceMentorListTestTags.list.lazyList,
    isScrollable = false,
)
```

https://github.com/KakaoCup/Compose/blob/master/compose/src/main/kotlin/io/github/kakaocup/compose/node/element/list/KListNode.kt

Новое API описания элемента списка

```
private class SimpleItem : KListItemNode<SimpleItem>() {

    val title by lazy {
        KNode(GalleryMagritteTestTags.lazyList.itemTitle)
    }

    val subtitle by lazy {
        KNode {
            hasTestTag(GalleryMagritteTestTags.lazyList.itemSubtitle)
        }
    }
}
```

Раздельное API для common и typed элементов

```
fun TestContext<*>.performClickOnItemWithTitle(text: String) {
    step("Клик на абстрактный элемент списка с заголовком (text: '$text')") {
        screenList.itemWith(
            { hasAnyDescendant(SemanticsMatcher.hasText(text)) }
        ) {
            performClick()
        }
    }
}

fun TestContext<*>.performClickOnItemWithTitle(text: String) {
    step("Клик на абстрактный элемент списка с заголовком (text: '$text')") {
        screenList.childWith<SimpleItem>(
            { hasAnyDescendant(SemanticsMatcher.hasText(text)) }
        ) {
            performClick()
        }

    }
}
```

Если не важно, что внутри элемента списка

```
fun TestContext<*>.performClickOnItemWithTitle(text: String) {
    step("Клик на абстрактный элемент списка с заголовком (text: '$text')") {
        screenList.itemWith(
            { hasAnyDescendant(SemanticsMatcher.hasText(text)) }
        ) {
            performClick()
        }
    }
}

fun TestContext<*>.performClickOnItemWithTitle(text: String) {
    step("Клик на абстрактный элемент списка с заголовком (text: '$text')") {
        screenList.childWith<SimpleItem>(
            { hasAnyDescendant(SemanticsMatcher.hasText(text)) }
        ) {
            performClick()
        }

    }
}
```

Если важно, что внутри элемента списка

```
fun TestContext<*>.performClickOnItemWithTitle(text: String) {
    step("Клик на абстрактный элемент списка с заголовком (text: '$text')") {
        screenList.itemWith(
            { hasAnyDescendant(SemanticsMatcher.hasText(text)) }
        ) {
            performClick()
        }
    }
}

fun TestContext<*>.performClickOnItemWithTitle(text: String) {
    step("Клик на абстрактный элемент списка с заголовком (text: '$text')") {
        screenList.childWith<SimpleItem>(
            { hasAnyDescendant(SemanticsMatcher.hasText(text)) }
        ) {
            performClick()
        }

    }
}
```

Profit!

1. Всегда требуется указывать типы элементов списка
2. Неконсистентное API работы с элементами
3. Конструктор элемента списка вербозен
4. KLazyListNode не подходит для not scrollable списков

Какие видим проблемы в существующем коде

1	Нужно везде пробрасывать ComposeTestRule	Ввели контейнер для хранения ComposeTestRule между тестами
2	По умолчанию используется merged-дерево семантики	Всегда работаем с unmerged-деревом семантики
3	Непонятно как работать с компонентами дизайн-системы	Для каждого компонента пишем отдельные KNode (мини Page Object)
4	Неудобное API для работы с Lazy-списками в Kakao	Подправили использование API, дописав немного функций и классов
5	**Неудобное API для работы с обычными списками в Kakao**	**Создали KListNode для тестирования любых типов списков**

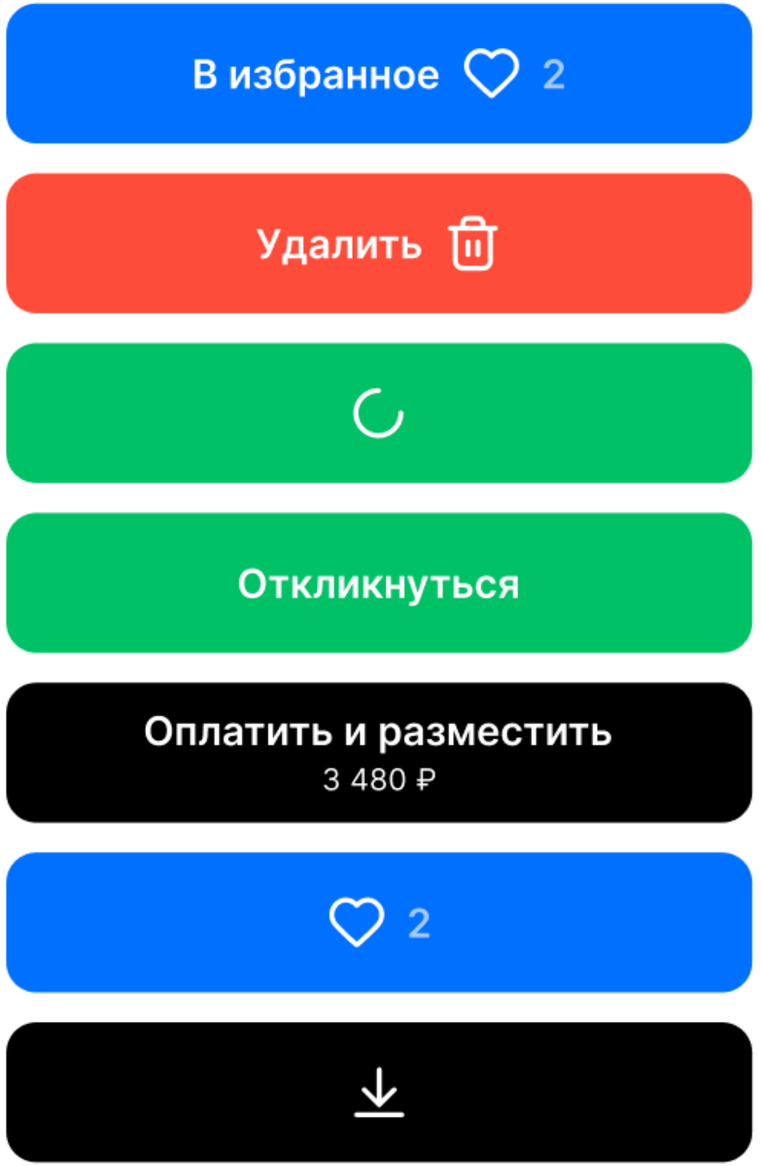
Akt III

Tips & tricks

Подводные камни тестирования Compose

1. Сравнение цвета background

Есть кнопки — хотим в тестах проверять цвет bg



Общий подход к такому сравнению — семантика

```
/**
 * Цвет background компонента.
 */
val BackgroundColor = SemanticsPropertyKey<Int>("BackgroundColor")

/**
 * Позволяет использовать getter и setter [BackgroundColor]
 * внутри блока [Modifier.semantics].
 */
@get:ColorInt
var SemanticsPropertyReceiver.backgroundColor by BackgroundColor
```

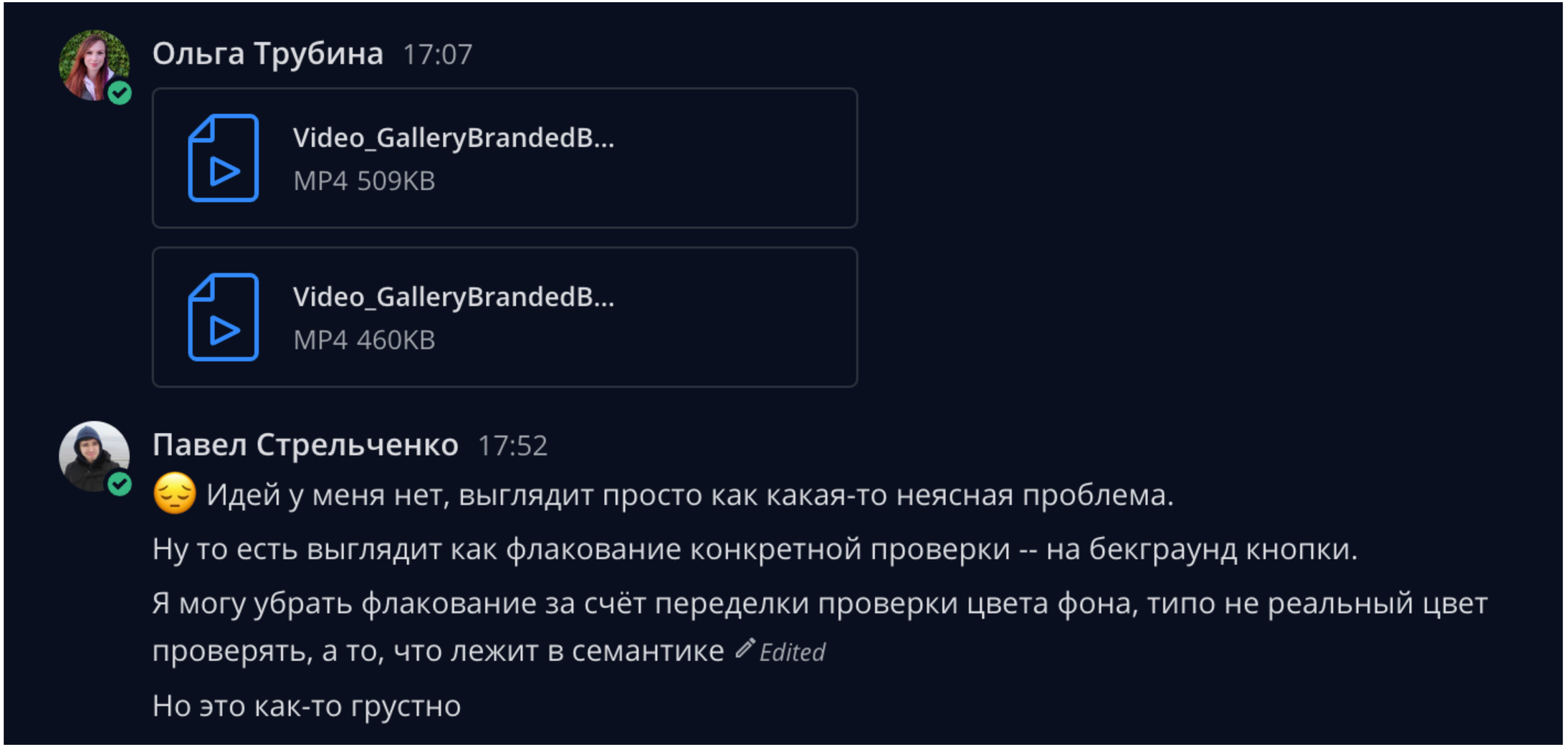
Почему бы не брать квадратик с пикселями?

```
fun assertBackgroundColor(color: Color) {
    delegate.check(NodeAssertions.ComposeBaseAssertionType.ASSERT) {
        val pixelsArray = IntArray(25)

        this.captureToImage()
            .readPixels(
                buffer = pixelsArray,
                startY = 5, startX = 5,
                width = 5, height = 5,
            )

        pixelsArray.forEachIndexed { _, pixel →
            check(pixel == color.toArgb())
        }
    }
```

Вводим процент fault tolerance

```
fun assertBackgroundColor(color: Color, faultTolerance: Float = 0.05f) {
    delegate.check(NodeAssertions.ComposeBaseAssertionType.ASSERT) {
        // ...

        val errorPixels = mutableListOf<Int>()
        pixelsArray.forEachIndexed { index, pixel ->
            if (pixel != color.toArgb()) {
                errorPixels.add(index)
            }
        }

        val errorPercent = calculateErrorPercent(errorPixels, pixelsArray)

        check(errorPercent < faultTolerance) {
            "Background color is not within tolerance! " +
                "[errorPercent: $errorPercent %, expectedColor: $
```

Но это не помогает ;(



Подводные камни тестирования Compose

1. Сравнение цвета background → Надёжнее всего через семантическое свойство

Подводные камни тестирования Compose

1. Сравнение цвета background → Надёжнее всего через семантическое свойство

2. **Сравнение картинок**

Придётся вводить семантическое свойство

```
@Composable
fun MagritteIcon(
    src: IconRes,
    modifier: Modifier = Modifier,
    tint: Color? = null,
    contentDescription: String? = null,
) {
    Image(
        painter = painterResource(id = src.id),
        colorFilter = tint?.let(ColorFilter::tint),
        contentDescription = contentDescription,
        modifier = modifier
            .semantics {
                iconRes = src.id
                iconTint = tint
            },
    )
```

Придётся вводить семантическое свойство

```
@Composable
fun MagritteIcon(
    src: IconRes,
    modifier: Modifier = Modifier,
    tint: Color? = null,
    contentDescription: String? = null,
) {
    Image(
        painter = painterResource(id = src.id),
        colorFilter = tint?.let(ColorFilter::tint),
        contentDescription = contentDescription,
        modifier = modifier
            .semantics {
                iconRes = src.id
                iconTint = tint
            },
    )
```

Подводные камни тестирования Compose

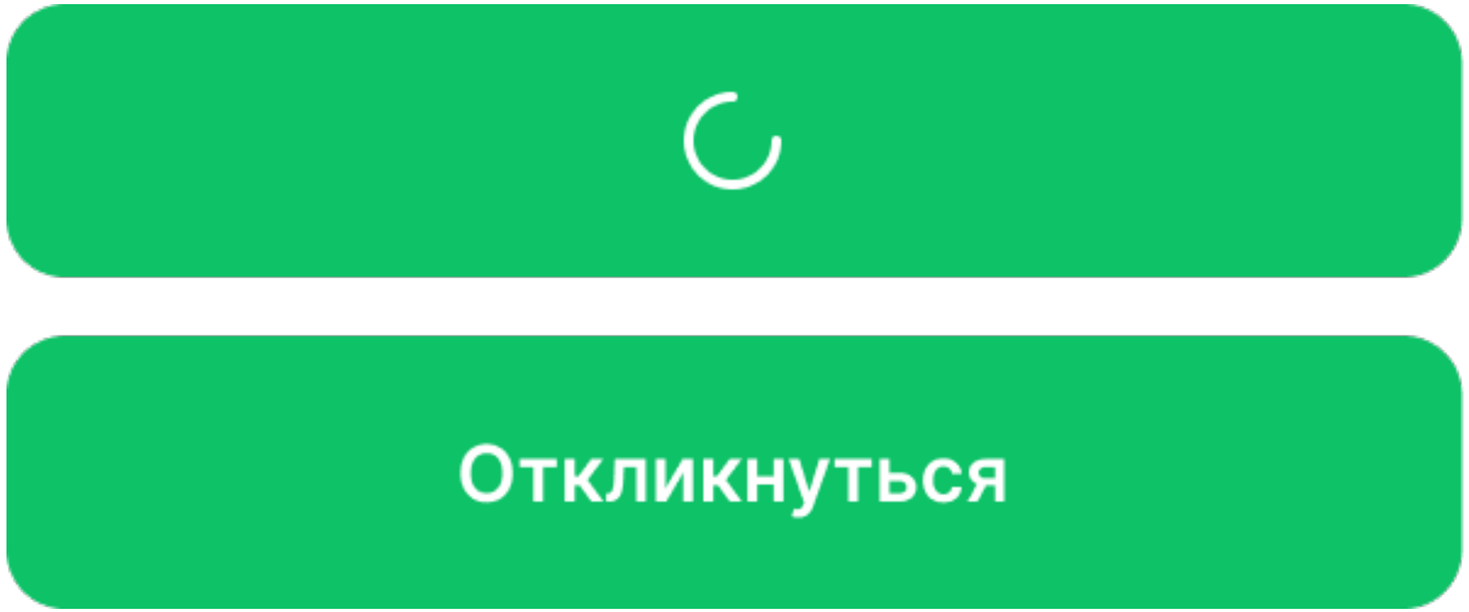
1. Сравнение цвета background → Надёжнее всего через семантическое свойство

2. **Сравнение картинок** → **Придётся делать через семантическое свойство**

Подводные камни тестирования Compose

1. Сравнение цвета background → Надёжнее всего через семантическое свойство
2. Сравнение картинок → Придётся делать через семантическое свойство
3. **Прозрачность и assertIsNotDisplayed**

Есть кнопка с состоянием Loading



Реализовали через graphicsLayer.alpha

```
val contentAlpha by animateFloatAsState(
    label = "ButtonContentAlpha",
    targetValue = if (contentState.loading) 0f else 1f
)

Row(
    modifier = Modifier
        .graphicsLayer {
            alpha = contentAlpha
        }
) {
    // Внутренний контент кнопки БЕЗ Loader
}
```

Проверяем видимость Loader в тесте...

```
class KButtonNode {
    /* ... */

    fun TestContext<*>.checkIsLoading() {
        step("Проверяем, что на кнопке показывается лоадер") {
            loader.assertIsDisplayed()

            // При отображении загрузки мы не убираем из иерархии
            // кнопки надписи или иконку, только скрываем. Поэтому
            // проверяем только то,
            // что контейнер контента не отображается на экране.
            content.assertIsNotDisplayed()
        }
    }
```

Но assertIsNotDisplayed падал с исключением

ru.hh.android.magritte_gallery.ui_test.tests.components.common.GalleryButtonScreenTest.test

java.lang.AssertionError: Assert failed: The component is displayed!
Failed on Pixel_4_API_34

Logs Device Info

```
08-23 14:47:01.009  9827  9851 E TestRunner:     at android.app
 .Instrumentation$InstrumentationThread.run(Instrumentation.java:2402)
08-23 14:47:01.009  9827  9851 E TestRunner: ----- end exception -----
08-23 14:47:01.020  9827  9851 I TestRunner: finished: test(ru.hh.android.magritte_gallery
 .ui_test.tests.components.common.GalleryButtonScreenTest)

java.lang.AssertionError Create breakpoint : Assert failed: The component is displayed!
at androidx.compose.ui.test.AssertionsKt.assertIsNotDisplayed(Assertions.kt:46)
at io.github.kakaocup.compose.node.assertion.NodeAssertions$assertIsNotDisplayed$1.invoke
 (NodeAssertions.kt:22)
at io.github.kakaocup.compose.node.assertion.NodeAssertions$assertIsNotDisplayed$1.invoke
 (NodeAssertions.kt:22)
at io.github.kakaocup.compose.intercept.operation
```

Переделали способ скрытия контента

```
fun Modifier.visible(visible: Boolean): Modifier {
    return if (visible) this else this.then(Invisible)
}

private object Invisible : LayoutModifier {

    override fun MeasureScope.measure(
        measurable: Measurable,
        constraints: Constraints,
    ): MeasureResult {
        val placeable = measurable.measure(constraints)
        return layout(placeable.width, placeable.height) {}
    }

}
```

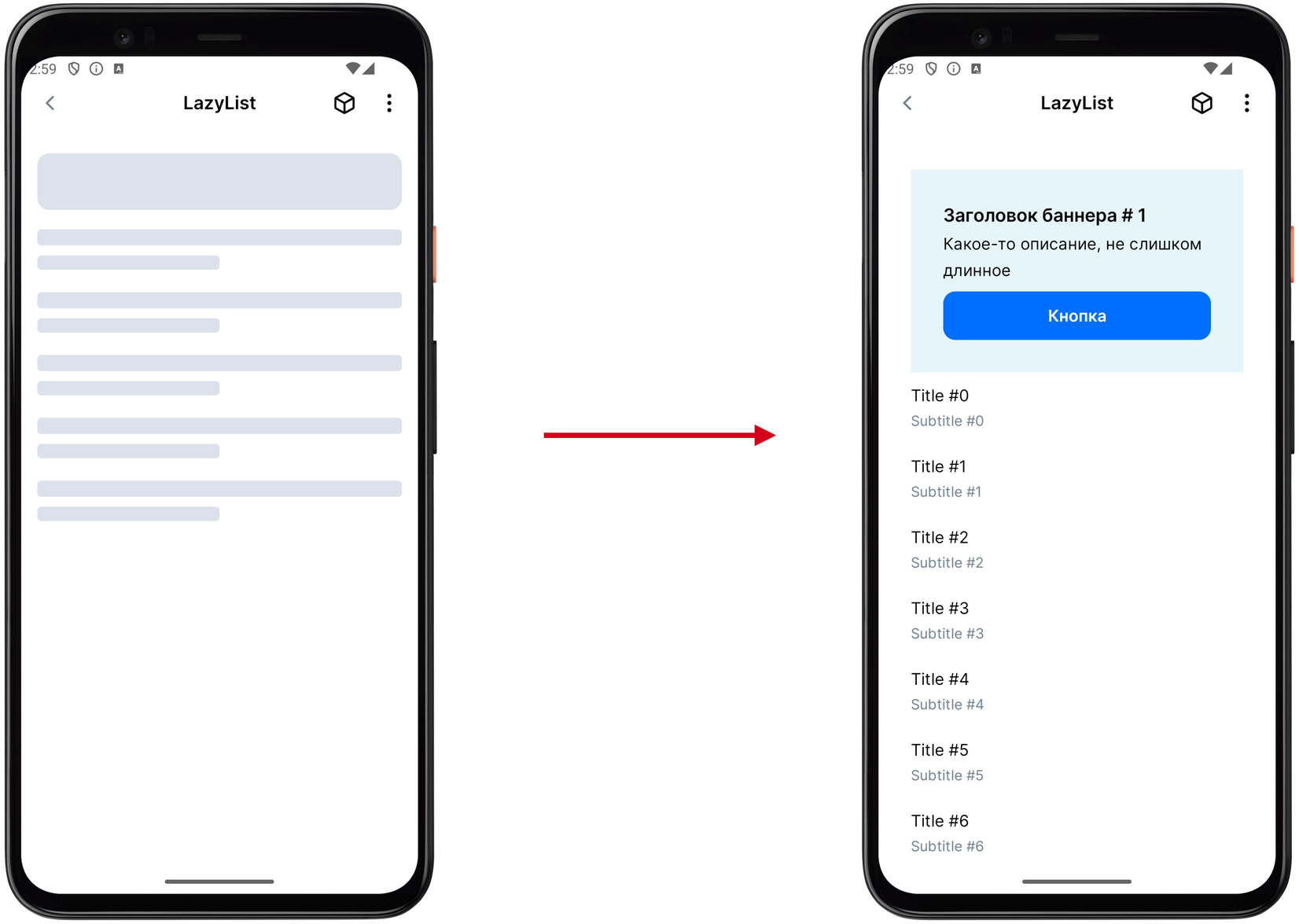
Подводные камни тестирования Compose

1. Сравнение цвета background → Надёжнее всего через семантическое свойство
2. Сравнение картинок → Придётся делать через семантическое свойство
3. **Прозрачность и assertIsNotDisplayed** → **Modifier.visible через Modifier.layout — рабочий вариант**

Подводные камни тестирования Compose

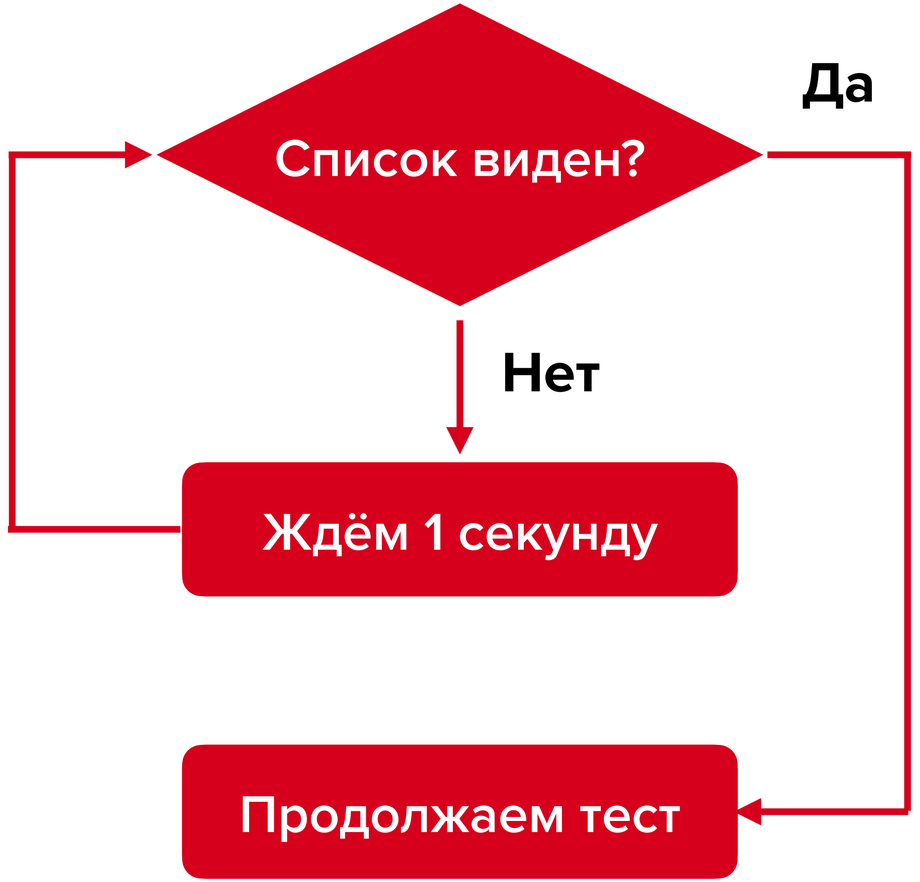
1	Сравнение цвета background	Надёжнее всего через семантическое свойство
2	Сравнение картинок	Придётся делать через семантическое свойство
3	Прозрачность и assertIsNotDisplayed	Modifier.visible через Modifier.layout — рабочий вариант
4	**Синхронизация виртуальных часов**	

Есть экран со списком и состоянием загрузки



У нас же есть flaky-safety механизм в Kaspresso!

```
private val screenList = KListNode {
    hasTestTag("lazy_list")
}


@Test
fun myTest() = run {
    // Специально триггерим flaky-safety
    screenList {
        assertIsDisplayed()
    }
}
```



Но тест всё равно упал!

ru.hh.android.magritte_gallery.ui_test.tests.components.other.GalleryLazyListScreenTest.test

java.lang.AssertionError: Assert failed: The component is not displayed!
Failed on Pixel_4_API_34

Logs | Device Info

```
java.lang.AssertionError Create breakpoint : Assert failed: The component is not displayed!
at androidx.compose.ui.test.AssertionsKt.assertIsDisplayed(Assertions.kt:34)
at io.github.kakaocup.compose.node.assertion.NodeAssertions$assertIsDisplayed$1.invoke(NodeAssertions
 .kt:13)
at io.github.kakaocup.compose.node.assertion.NodeAssertions$assertIsDisplayed$1.invoke(NodeAssertions
 .kt:13)
at io.github.kakaocup.compose.intercept.operation.ComposeOperationImplsKt$produceComposeAssertion$1
 .execute(ComposeOperationImpls.kt:29)
at io.github.kakaocup.compose.intercept.operation.ComposeOperationImplsKt$produceComposeAssertion$1
 .execute(ComposeOperationImpls.kt:26)
```

```
at com.kaspersky.components.composesupport.interceptors.tolibrary
 .ComposeSemanticsInterceptor$interceptCheck$2$1.invoke(ComposeSemanticsInterceptor.kt:33)
at com.kaspersky.components.composesupport.interceptors.tolibrary
 .ComposeSemanticsInterceptor$interceptCheck$2$1.invoke(ComposeSemanticsInterceptor.kt:33)
at com.kaspersky.kaspresso.flakysafety.algorithm.FlakySafetyAlgorithm.invokeFlakySafely
 (FlakySafetyAlgorithm.kt:32)
at com.kaspersky.kaspresso.flakysafety.algorithm.FlakySafetyAlgorithm.invokeFlakySafely$default
 (FlakySafetyAlgorithm.kt:22)
at com.kaspersky.kaspresso.flakysafety.FlakySafetyProviderSimpleImpl.flakySafely
 (FlakySafetyProviderSimpleImpl.kt:27)
```

Существуют «виртуальные часы» Compose UI

Manual synchronization

In certain cases, you have to synchronize the Compose UI with other parts of your test or the app you're testing.

The `waitForIdle()` function waits for Compose to be idle, but the function depends on the `autoAdvance` property:

```
composeTestRule.mainClock.autoAdvance = true // Default
composeTestRule.waitForIdle() // Advances the clock until Compose is idle.

composeTestRule.mainClock.autoAdvance = false
composeTestRule.waitForIdle() // Only waits for idling resources to become idle.
```

Note that in both cases, `waitForIdle()` also waits for pending draw and layout passes.

Also, you can advance the clock until a certain condition is met with `advanceTimeUntil()`.

```
composeTestRule.mainClock.advanceTimeUntil(timeoutMs) { condition }
```

Подменяем реализацию flaky-safety

```
return Kaspresso.Builder.withForcedAllureSupport().apply {
    addComposeSupport(
        lateComposeCustomize = { composeConfig →
            // Переопределяем логику flaky safety,
            // чтобы поддержать тестирование Compose-экранов с анимациями.
            val standardInterceptors = composeConfig.semanticsBehaviorInterceptors.filter {
                it !is FlakySafeSemanticsBehaviorInterceptor
            }

            composeConfig.semanticsBehaviorInterceptors = standardInterceptors.toMutableList().apply {
                this += HHFlakySafeSemanticsBehaviorInterceptor(flakySafetyParams, libLogger)
            }
        }
    )
}
```

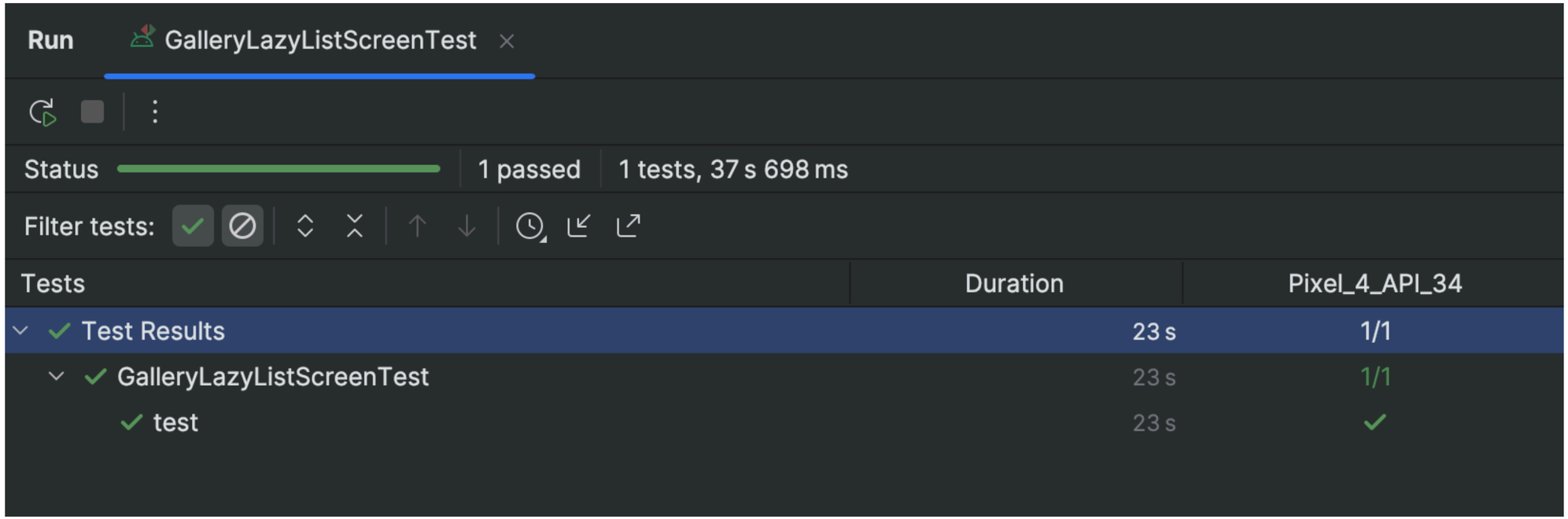
В дебрях реализации flaky-safety interceptor

```
do {
    try {
        return action.invoke()
    } catch (error: Throwable) {
        if (error.isAllowed(params.allowedExceptions)) {
            cachedError = error
            lock.withLock {
                Timer().schedule(params.intervalMs) {
                    // Специально проматываем время в рамках теста Compose-экрана.
                    ComposeRuleContainer.rule?.mainClock?.advanceTimeBy(params.intervalMs)

                    lock.withLock { condition.signalAll() }
                }
                condition.await()
            }
        } else {
            throw error
        }
    }
} while (System.currentTimeMillis() - startTime ⩽ params.timeoutMs)
```

Специально проматываем время

```
do {
    try {
        return action.invoke()
    } catch (error: Throwable) {
        if (error.isAllowed(params.allowedExceptions)) {
            cachedError = error
            lock.withLock {
                Timer().schedule(params.intervalMs) {
                    // Специально проматываем время в рамках теста Compose-экрана.
                    ComposeRuleContainer.rule?.mainClock?.advanceTimeBy(params.intervalMs)

                    lock.withLock { condition.signalAll() }
                }
                condition.await()
            }
        } else {
            throw error
        }
    }
} while (System.currentTimeMillis() - startTime ⩽ params.timeoutMs)
```

Механизм flaky-safety спасён



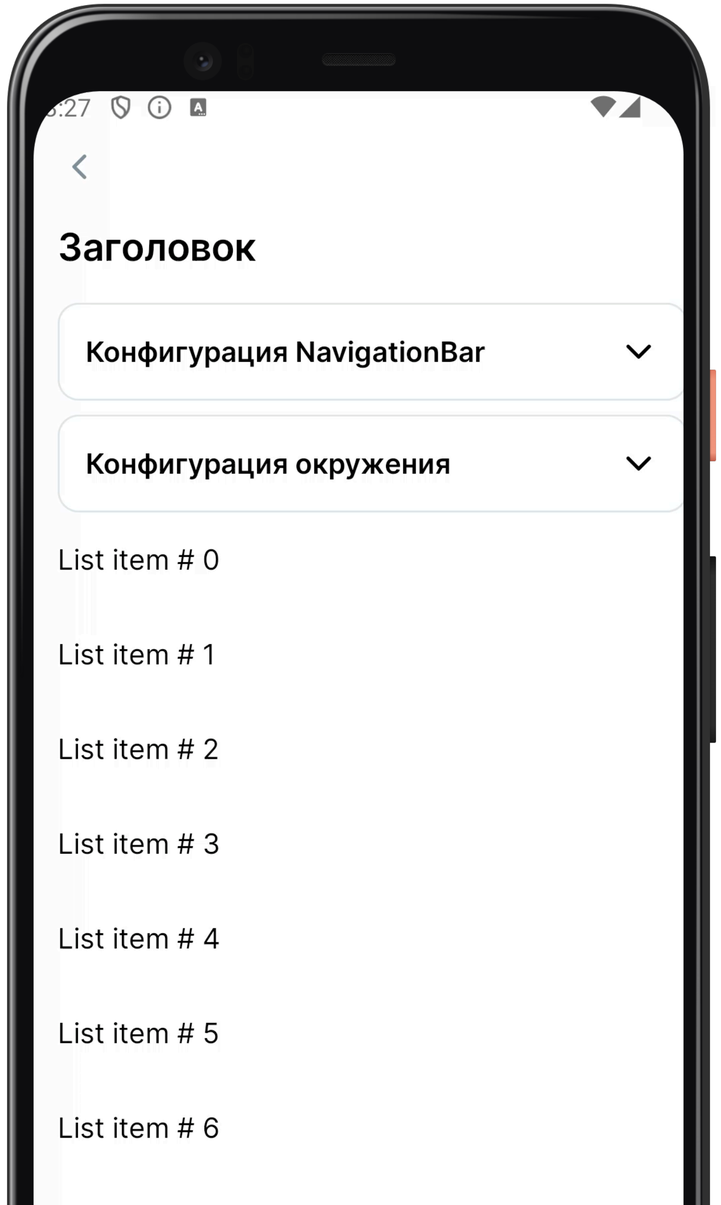
Подводные камни тестирования Compose

1	Сравнение цвета background	Надёжнее всего через семантическое свойство
2	Сравнение картинок	Придётся делать через семантическое свойство
3	Прозрачность и assertIsNotDisplayed	Modifier.visible через Modifier.layout — рабочий вариант
4	**Синхронизация виртуальных часов**	**Для корректного пропуска анимаций пришлось подменять интерсептор**

Подводные камни тестирования Compose

1	Сравнение цвета background	Надёжнее всего через семантическое свойство
2	Сравнение картинок	Придётся делать через семантическое свойство
3	Прозрачность и assertIsNotDisplayed	Modifier.visible через Modifier.layout — рабочий вариант
4	Синхронизация виртуальных часов	Для корректного пропуска анимаций пришлось подменять интерсептор
5	**performScrollToNode не скроллит «по-настоящему»**	

Есть экран с Collapsing toolbar



Хочется написать тест на смену expanded

```
step("Проверяем состояние экрана после скролла списка для Collapsing NavBar")
    galleryNavigationBarComposePageObject.actions {
        performScrollToLastListItems()
    }

    galleryNavigationBarComposePageObject.checks {
        checkNavigationBarExpandedTitleIsNotDisplayed()
        checkNavigationBarExpandedSubtitleIsNotDisplayed()
        checkNavigationBarCollapsedTitle(DEFAULT_TITLE)
        checkNavigationBarCollapsedSubtitleIsNotDisplayed()
    }

    galleryNavigationBarComposePageObject.actions {
        performScrollToFirstListItems()
    }
```

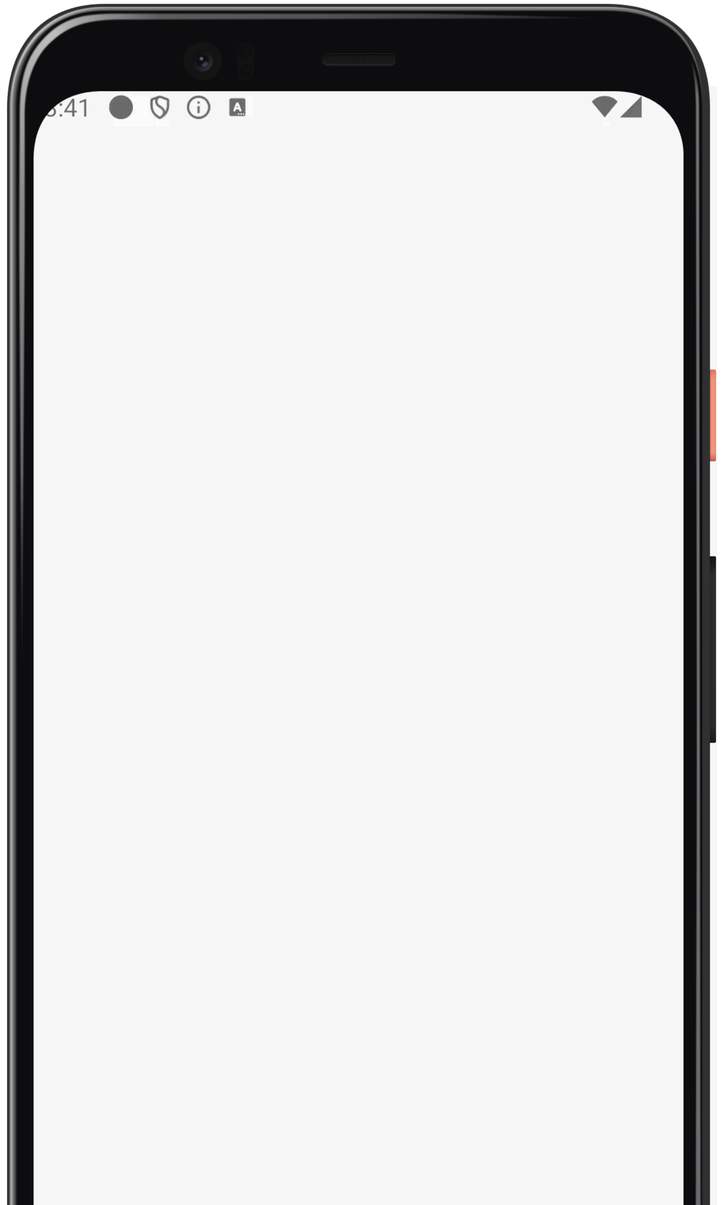
Скроллим вниз, а потом вверх

```
step("Проверяем состояние экрана после скролла списка для Collapsing NavBar")
    galleryNavigationBarComposePageObject.actions {
        performScrollToLastListItems()
    }

    galleryNavigationBarComposePageObject.checks {
        checkNavigationBarExpandedTitleIsNotDisplayed()
        checkNavigationBarExpandedSubtitleIsNotDisplayed()
        checkNavigationBarCollapsedTitle(DEFAULT_TITLE)
        checkNavigationBarCollapsedSubtitleIsNotDisplayed()
    }

    galleryNavigationBarComposePageObject.actions {
        performScrollToFirstListItems()
    }
```

Логичная реализация скролла

```
@OptIn(ExperimentalTestApi::class)
override fun TestContext<*>.performScrollToLastListItems() {
    step("Имитируем скролл к последним элементам списка") {
        mainLazyList.performScrollToNode(
            SemanticsMatcher.expectValue(CustomSemantics.ListItemIndex, 50)
        )
    }
}
```

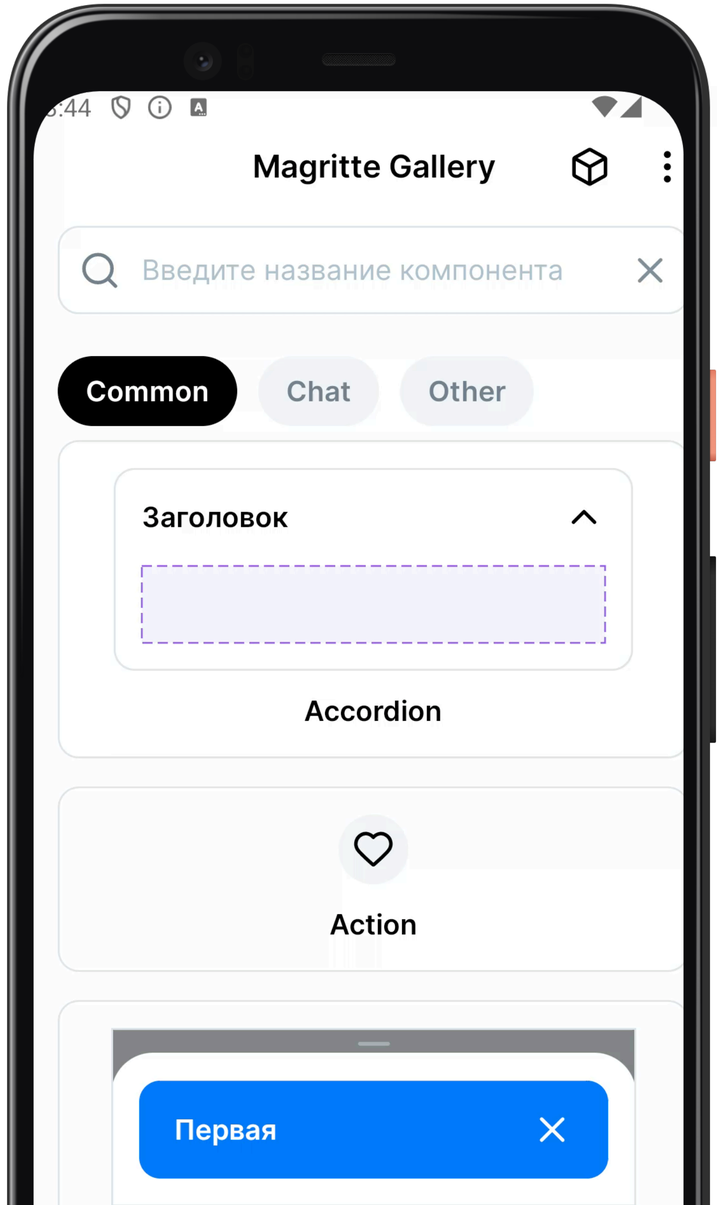
Скроллит, но не скроллит



А как правильно?

```
override fun TestContext<*>.performScrollToLastListItems() {
    step("Имитируем скролл через свайп к последним элементам списка") {
        mainLazyList.performTouchInput {
            swipeUp()
            swipeUp()
        }
    }
}
```

Теперь всё хорошо



Подводные камни тестирования Compose

1	Сравнение цвета background	Надёжнее всего через семантическое свойство
2	Сравнение картинок	Придётся делать через семантическое свойство
3	Прозрачность и assertIsNotDisplayed	Modifier.visible через Modifier.layout — рабочий вариант
4	Синхронизация виртуальных часов	Для корректного пропуска анимаций пришлось подменять интерсептор
5	**performScrollToNode не скроллит «по-настоящему»**	**Там где это нужно, можно пользоваться performTouchInput**

Акт IV

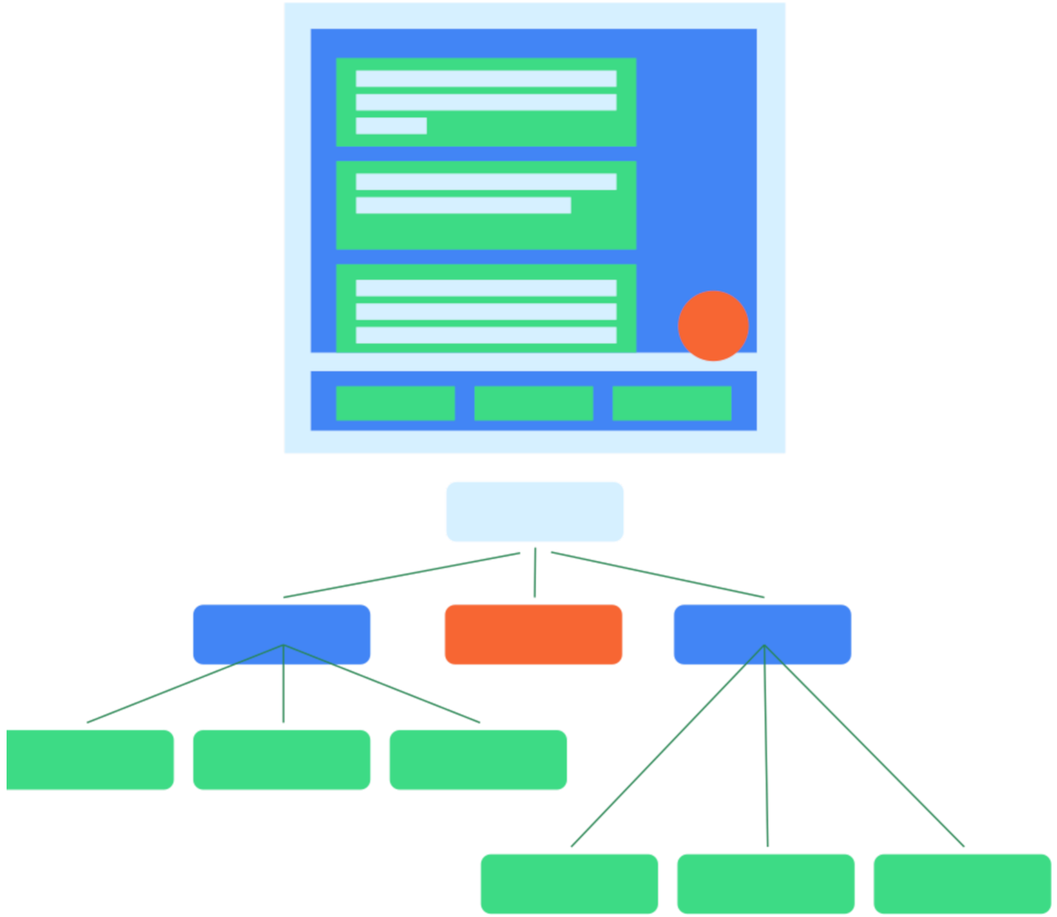
Бонус-секция

А можно ли отказаться от тестовых тегов?

Много тестовых тегов — грусть

Может попробуем опираться на семантику?

Merged Semantics	
GetTextLayout...	AccessibilityAction
OnClick	AccessibilityAction
Role	Button
Text	List[1]
[0]	Like
Semantics	
OnClick	AccessibilityAction
Role	Button



Вводим «тип компонента» в семантику

```
@Composable
fun MagritteIcon(
    src: IconRes,
    modifier: Modifier = Modifier,
    tint: Color? = null,
    contentDescription: String? = null,
) {
    Image(
        /* ... */
        modifier = modifier
            .semantics {
                iconRes = src.id
                iconTint = tint
                componentType = SemanticsComponentType.Icon
            },
    )
}
```

Тип компонента добавлен для компонентов ДС

```
@Composable
fun MagritteIcon(
    src: IconRes,
    modifier: Modifier = Modifier,
    tint: Color? = null,
    contentDescription: String? = null,
) {
    Image(
        /* ... */
        modifier = modifier
            .semantics {
                iconRes = src.id
                iconTint = tint
                componentType = SemanticsComponentType.Icon
            },
    )
}
```

Тогда искать узел в тесте можно вот так

```
private val myButton = KNode {
     hasComponentType(ComponentType.Button)
     hasAnyDescendantWithText("Текст на кнопке")
}

private val myButton = KButtonNode {
    hasAnyDescendantWithText("Текст на кнопке")
}
```

Существующий Role использовать не получится

```
@Immutable
@kotlin.jvm.JvmInline
value class Role private constructor(
    private val value: Int
) {
    companion object {
        val Button = Role(0)
        val Checkbox = Role(1)
        val Switch = Role(2)
        val RadioButton = Role(3)
        val Tab = Role(4)
        val Image = Role(5)
        val DropdownList = Role(6)
    }
}
```

За и против

1. Уходит явно «тестовый» код из описания экранов
2. Учимся взаимодействовать с accessibility
3. Сильно сложнее для понимания
4. Не получается уйти от тестовых тегов полностью

Пока что мы продолжаем использовать теги

podlodka

Android Crew

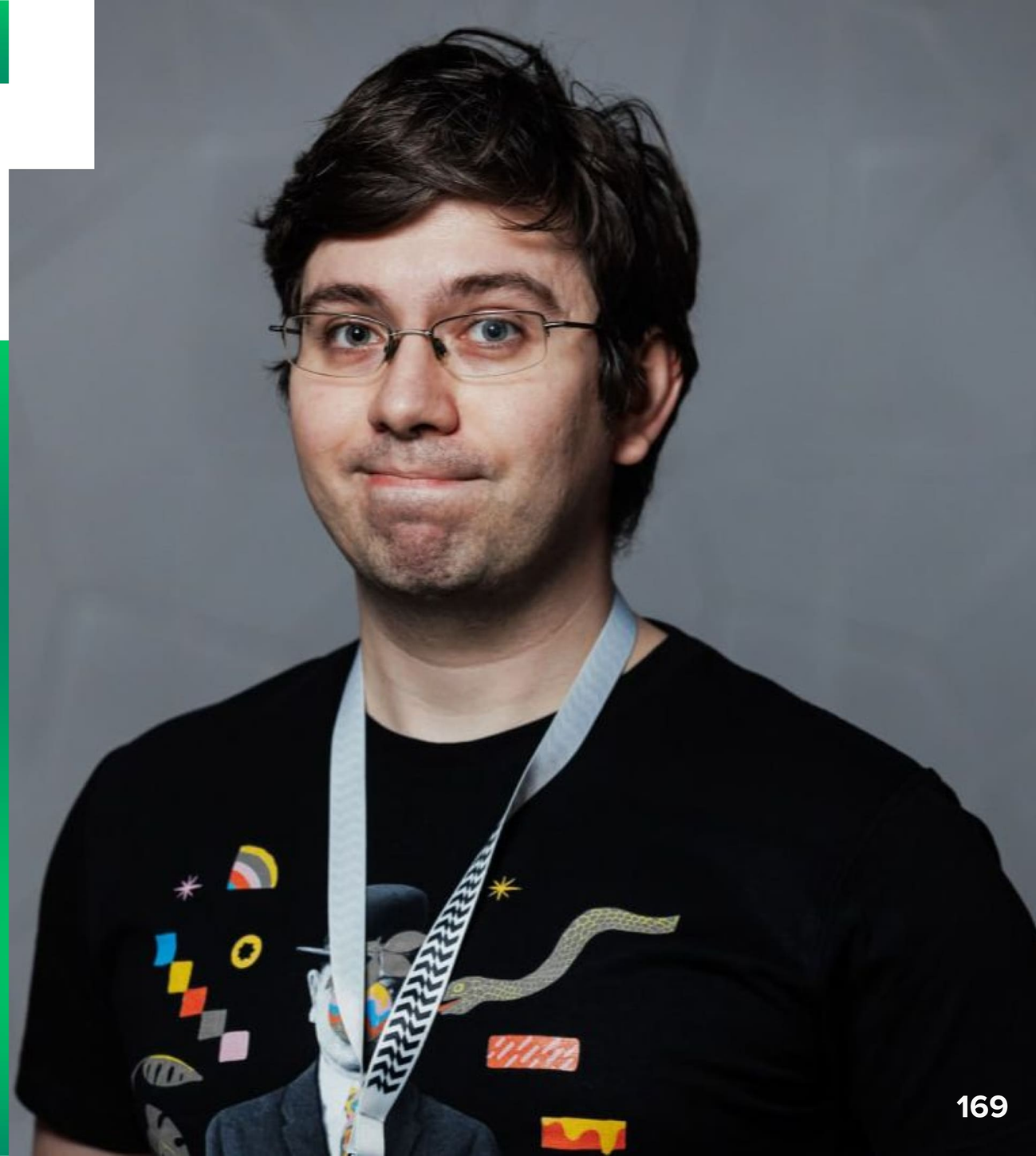
16 – 20 сентября

Автоматизация разработки

Павел Стрельченко

HeadHunter

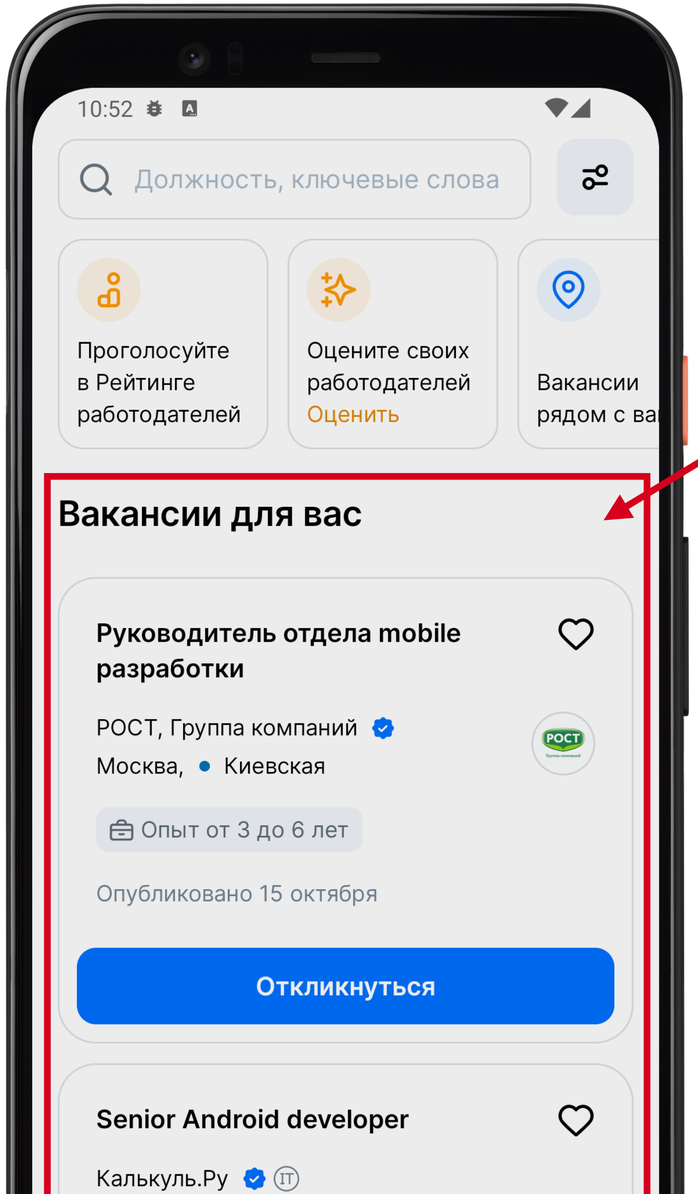
Воркшоп: Пришёл, увидел, наплагинил



Проблемы интеропа с View-миром

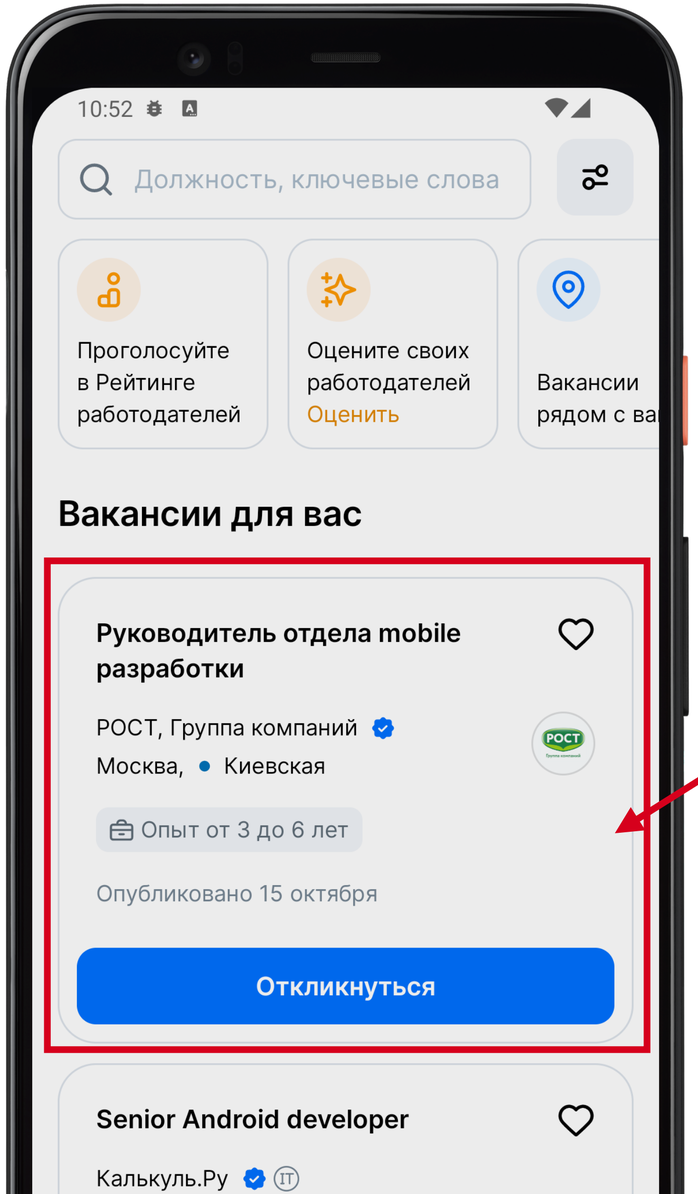
Редизайн не происходит мгновенно

В коде есть RecyclerView с Compose-элементами



Переиспользуемый на нескольких экранах RecyclerView

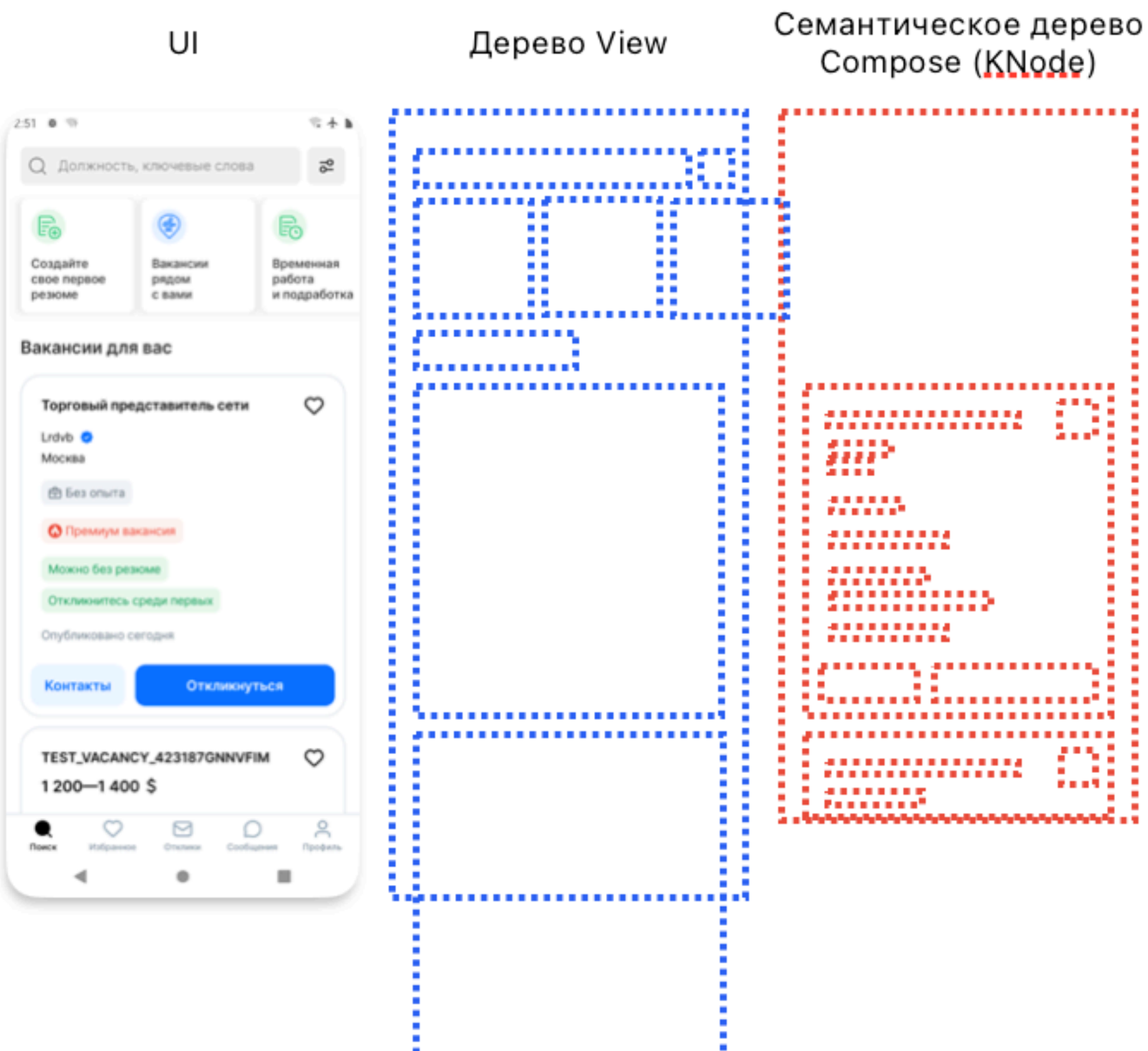
В коде есть RecyclerView с Compose-элементами



Compose-ячейка

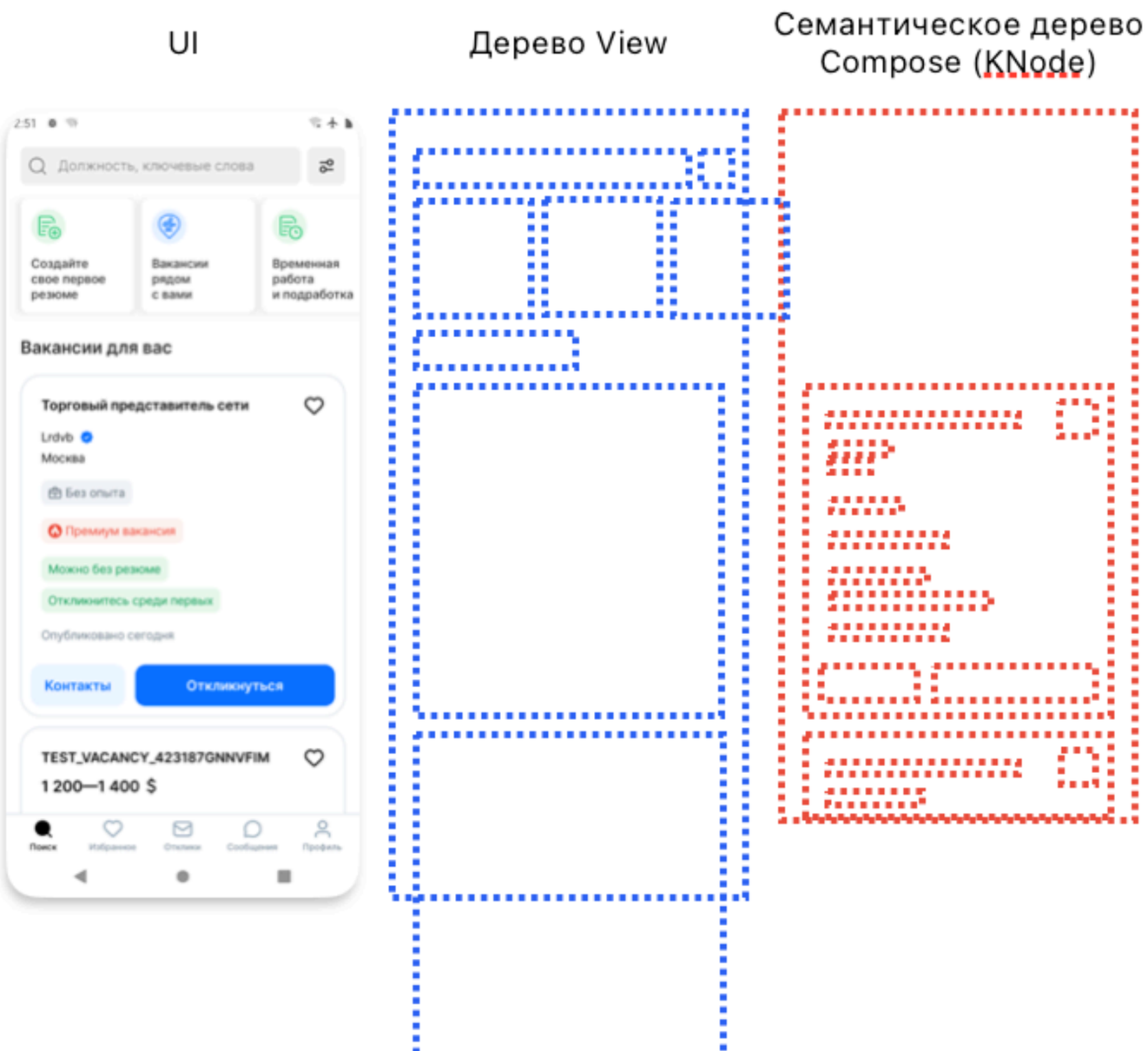
RecyclerView + элементы с Compose внутри

1. Дерево View не связано с Compose-деревом семантики



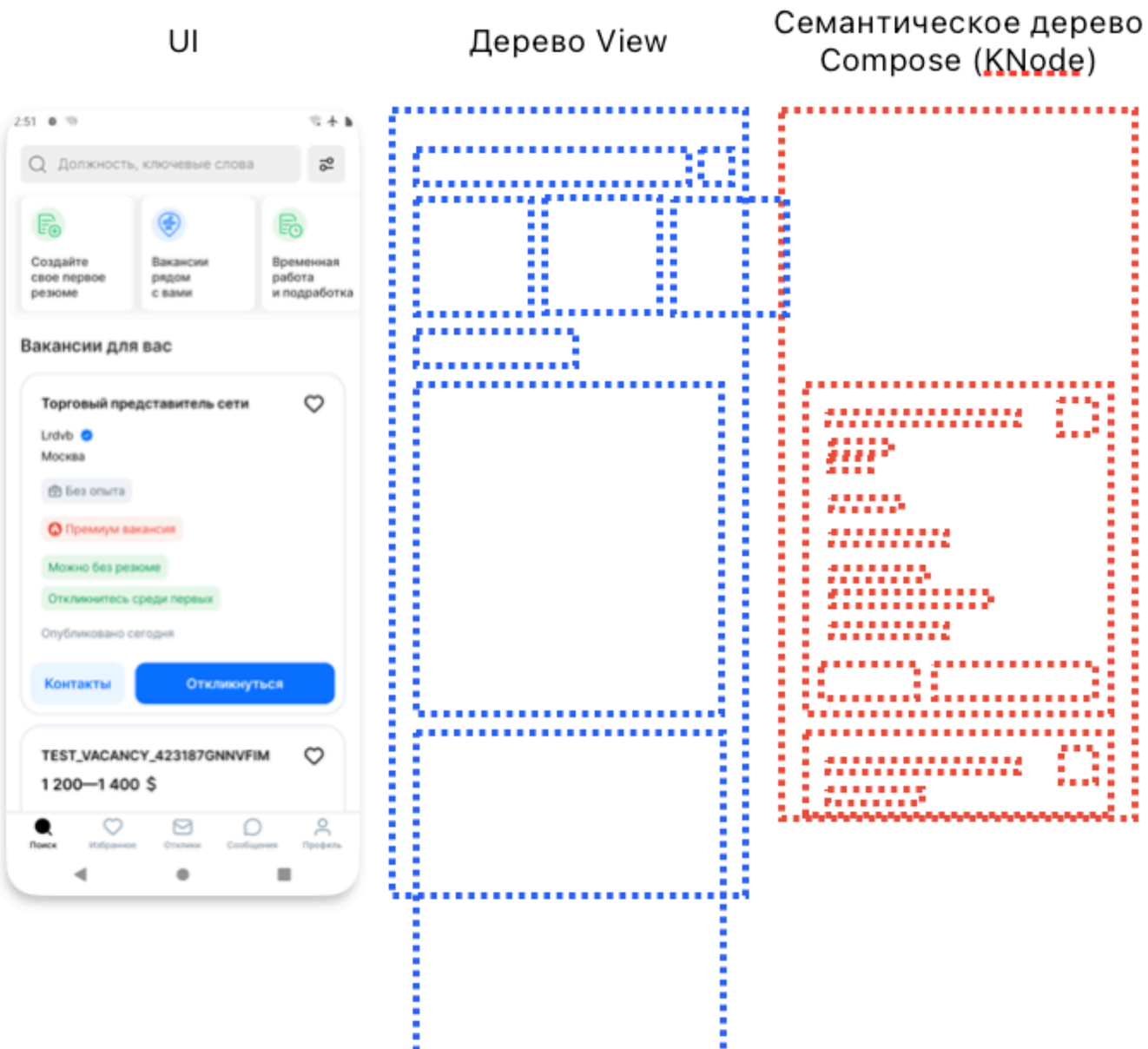
RecyclerView + элементы с Compose внутри

1. Дерево View не связано с Compose-деревом семантики
2. **Из KRecyclerView напрямую получить доступ нельзя**



RecyclerView + элементы с Compose внутри

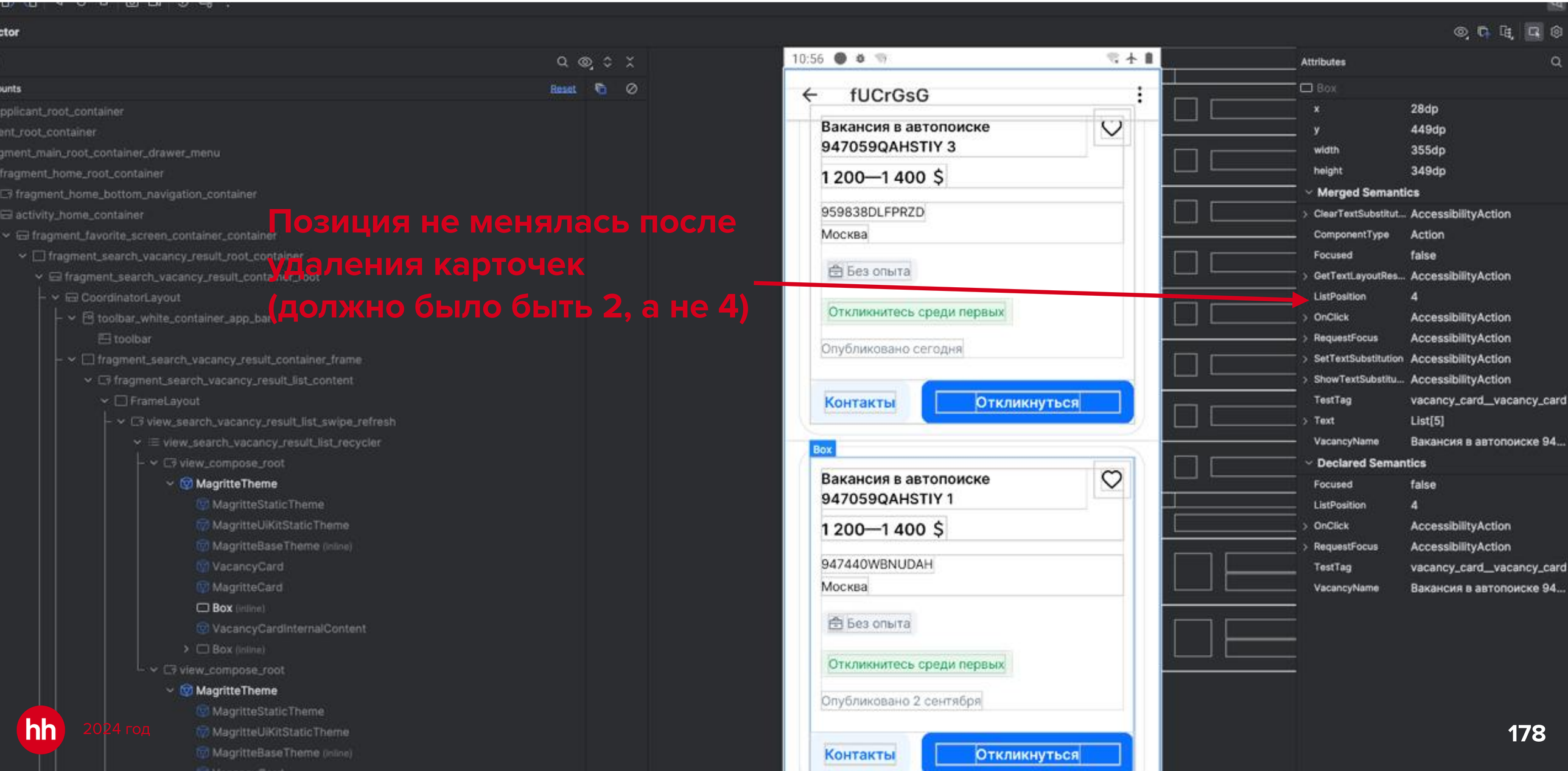
1. Дерево View не связано с Compose-деревом семантики
2. Из KRecyclerView напрямую получить доступ нельзя
3. **Скроллим до нужного элемента и ищем по семантике экрана**



Добавить позицию адаптера в Compose?..

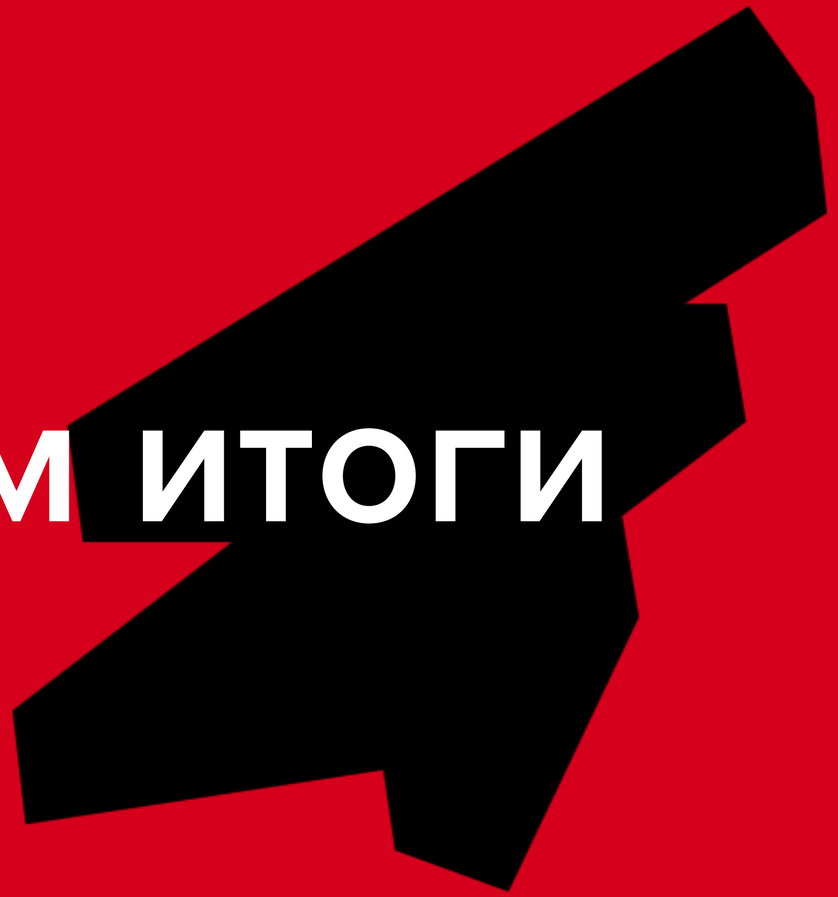
```
        onBindView(composeView, viewHolder.absoluteAdapterPosition)
    }

    Alexander +2 *
    private fun onBindView(view: ComposeView, listPosition: Int) {
        view.setContent {
            MagritteTheme {
                if (brandCoverUiModel != null) {
                    BrandVacancyCard(
                        modifier = Modifier
                            .paddingIf(!vacancyCardUiModel.withoutMargin)
                            .semanticsListPosition(listPosition),
                        vacancyCardUiModel = vacancyCardUiModel
```

Но позиция в семантике не обновлялась



Нужно будет проводить ресёрч решений

Подведём итоги



Итоги

UI-тесты для Compose-экранов возможны

Можно пользоваться стандартным фреймворком или библиотеками

API ваших тестов можно улучшать

Часть наших предложений по улучшениям API скоро войдут в Kaspresso и Kakao

Есть много «подводных камней»

Про часть из них вы теперь знаете =)

Спросите меня о чём-нибудь ;)

Что есть «из коробки»?

Расскажу про стандартный фреймворк тестирования Compose-экранов от Google

Улучшаем API

Покажу, как мы итеративно улучшали API UI-тестов в нашей кодовой базе на базе Kaspresso

Tips & tricks

Обсудим подводные камни в тестировании Compose-экранов и как их обойти

Автор прекрасных стикеров про Нюхлёра: https://www.instagram.com/maxhasky.art/