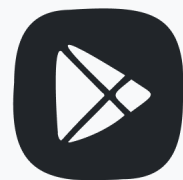


Живая картинка

анимации и эффекты
в кроссплатформенном
BDUI-движке

Телышев Алексей





более 50 млн

скачиваний в Google Play



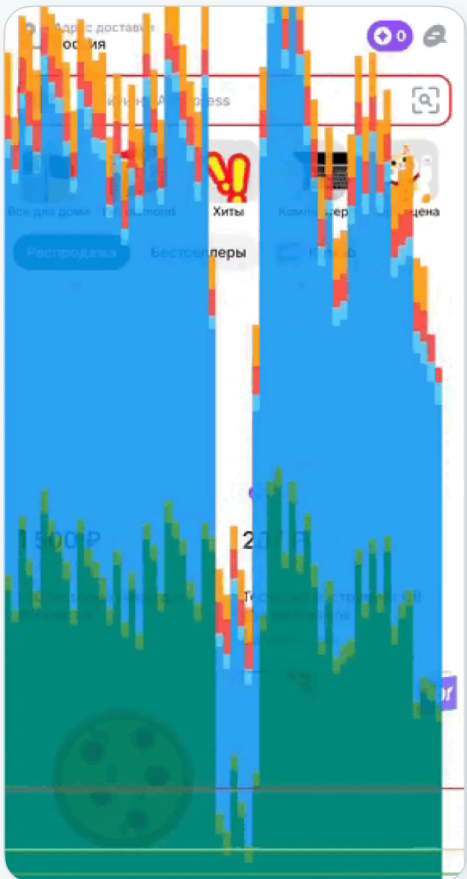
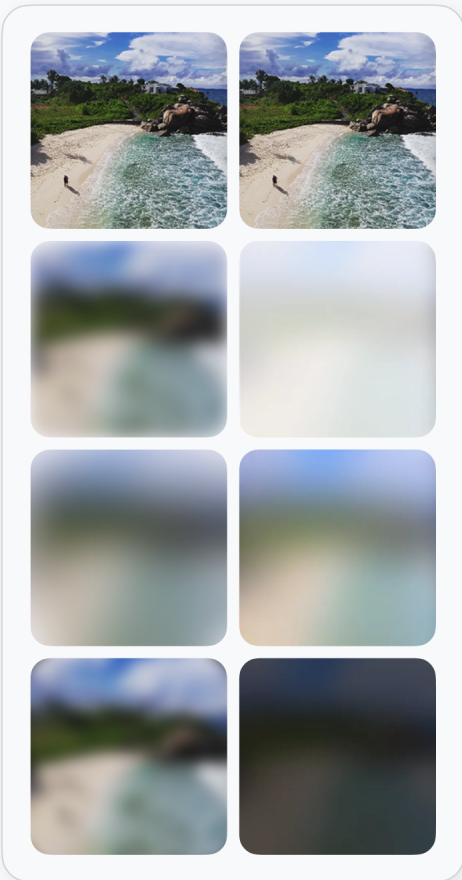
более 1 млн

отзывов в App Store

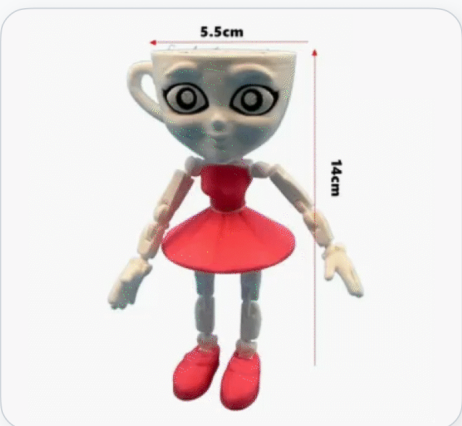


Aa Inter 40 525

● text/text_primary

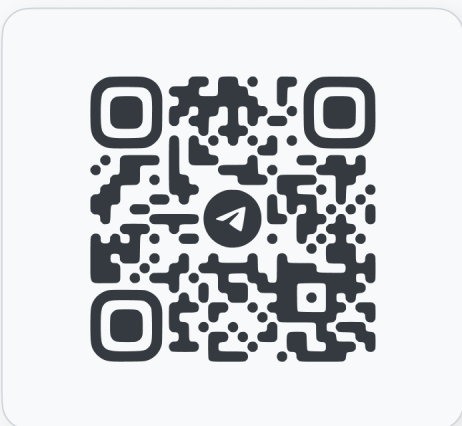


FUSION



```
func pause() {
  isPaused = true
  animationView?.pause()
}

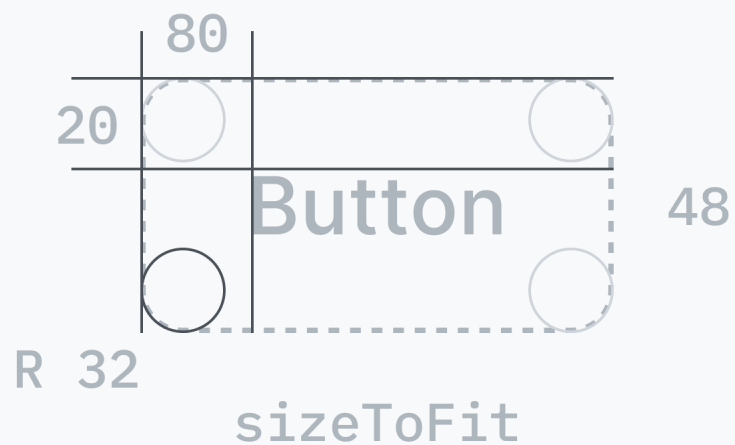
func resume() {
  isPaused = false
  play(fromProgress: minProgress, to: maxProgress)
}
```



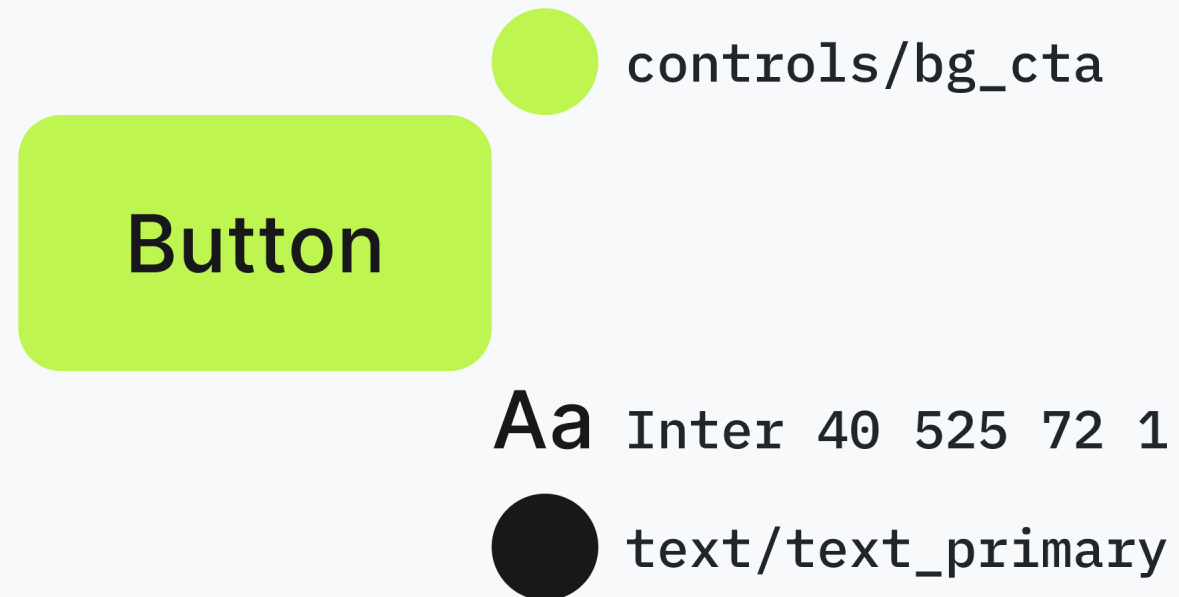
FUSION

Телышев Алексей

ATOM



Атрибуты



Выражения

Button

disabled



loading

Button

Initial

Button

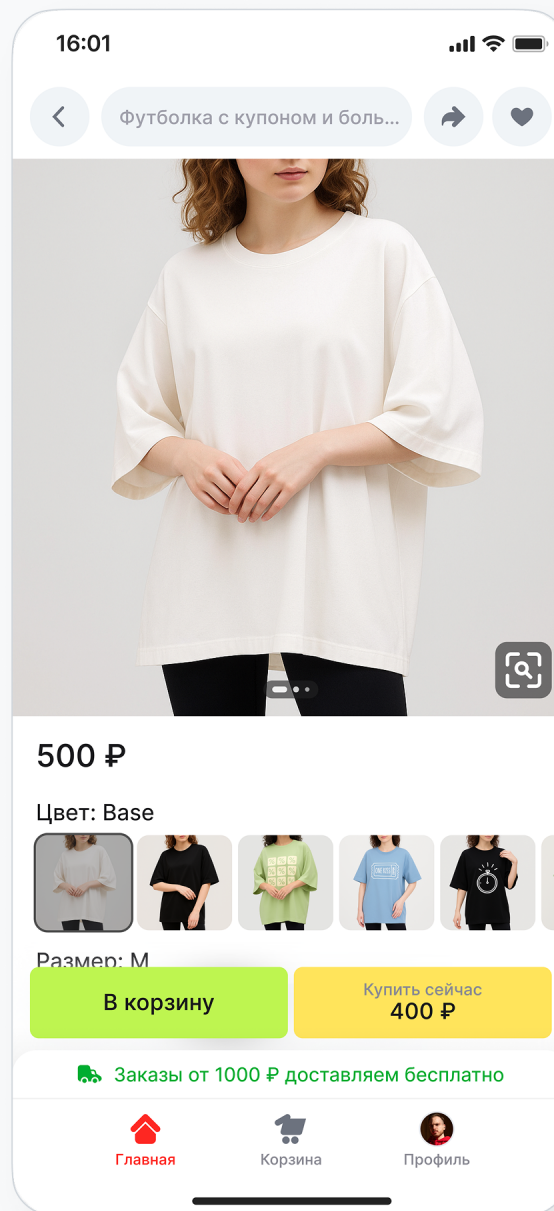
pressed

Молекула

В корзину

Купить сейчас

Шаблон



Телышев Алексей

Вращение

```
01 val AliExpress by molecule(Version(release = 1)) {  
02  
03     Image {  
04         rotation { 90.0.c }  
05     }  
06 }
```



ВРАЩАТЬ

Изменения iOS

```
01 view.frame = frame  
02 view.bounds.size = frame.size  
03 view.center = CGPoint(x: frame.midX, y: frame.midY)
```

Изменения iOS

```
01 child.frame.origin = origin
02 if child.transform == .identity {
03     return child.frame.origin = origin
04 }
05
06 let transformedBounds = child.bounds.applying(child.transform)
07 let center = CGPoint(x: origin.x + transformedBounds.width / 2, y: origin.y + transformedBounds.height / 2)
08 child.center = center
```

Изменения iOS

```
01  for child in self.fillParentChildren {
02      let frame = fill(child: child)
03      sendNonFatalLogIfNeeded(rect: frame, logPrefix: "fillParentChildren iteration")
04      child.frame = frame.isCalculableValues() ? frame : .zero
05      if frame.isCalculableValues() {
06          child.bounds.size = frame.size
07          child.center = CGPoint(x: frame.midX, y: frame.midY)
08      } else {
09          child.frame = .zero
10      }
11  }
```

Изменения iOS

```
01  import Foundation
02
03  extension CGAffineTransform {
04      static func rotatedBy(degrees: CGFloat) -> Self {
05          CGAffineTransform(rotationAngle: degrees.radians)
06      }
07  }
```

Изменения iOS

```
01  var radians: CGFloat {  
02      self * .pi / 180  
03  }
```

Изменения iOS

```
01  if node.viewAttributes.rotationIsChanged, let rotation = node.rotation.map({ CGFloat(truncating: $0) }) {
02      let transform: CGAffineTransform = .rotatedBy(degrees: rotation)
03      if view.transform != transform {
04          view.transform = transform
05      }
06  }
```

Изменения Android

```
01 if (rotation.isChanged()) {  
02     view.rotation = rotation.value?.toFloat() ?: 0f  
03 }
```

Таймер

```
01 state.counter.set(0.c)
02 launchTimer(
03     stepMillis = 100.c,
04     initialDelayMillis = 1000.c
05 ) {
06     state.counter.set(state.counter.plus(1.c))
07 }
```

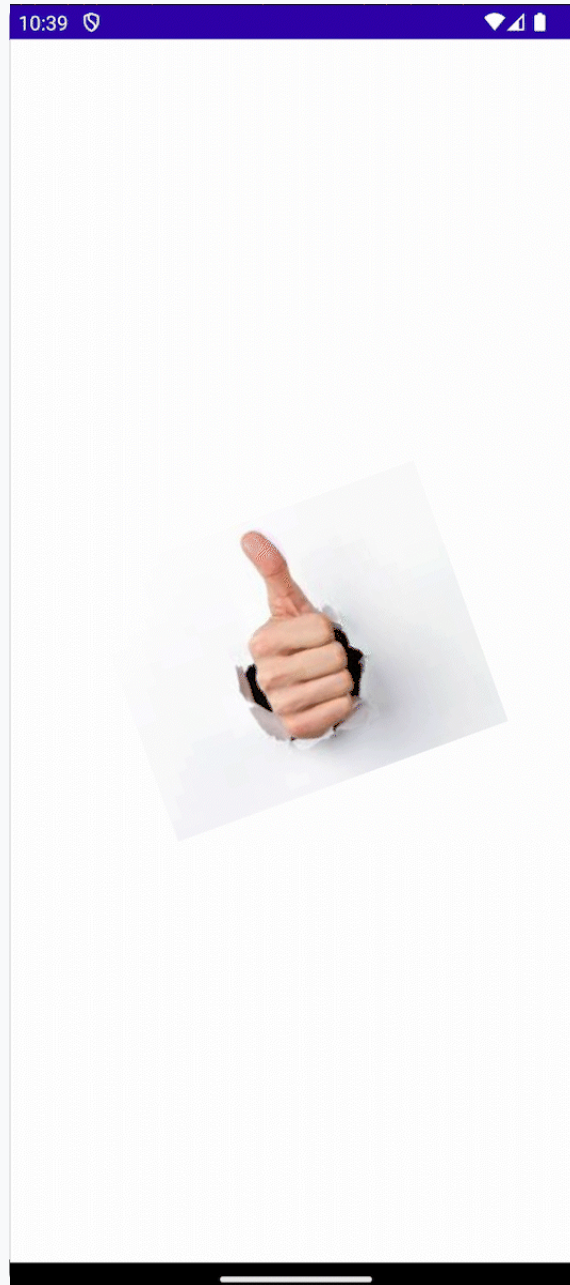

Fusion

```
01 fun FunctionScope.launchTimer(  
02     stepMillis: FInt,  
03     initialDelayMillis: FInt = 0.c,  
04     maxStepsCount: FInt = Long.MAX_VALUE.c,  
05     onResult: () -> FAny,  
06 ): FAny = anyFunction(  
07     id = BizFunctionIds.timer,  
08     hasSideEffect = false,  
09     stepMillis,  
10     executeBlock { onResult() },  
11     initialDelayMillis,  
12     maxStepsCount  
13 )
```

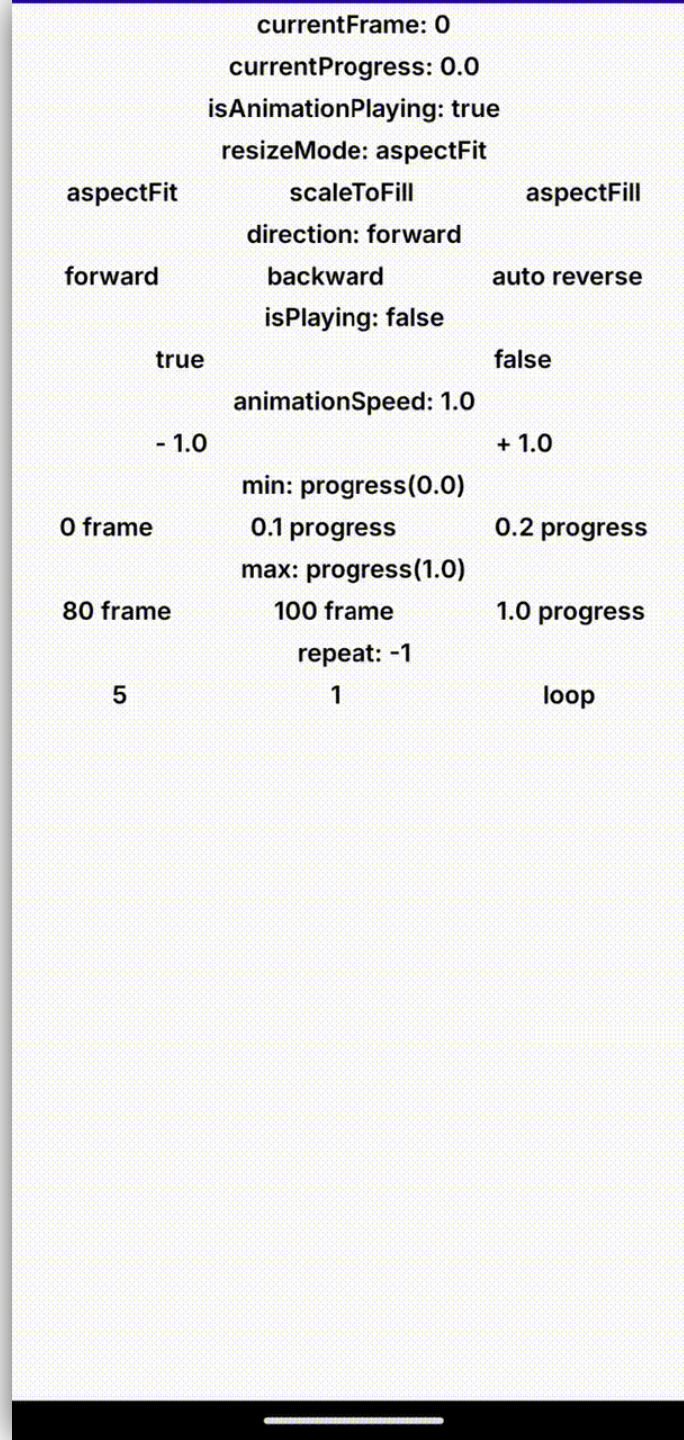
Fusion

```
01 internal object Timer : FusionFunction {
02     override val id = BizFunctionIds.timer
03     override val isPure: Boolean = true
04
05     override fun eval(
06         args: FusionFunction.Args,
07         context: FusionContext,
08         scope: FusionScope?
09     ): Any? {
10         val stepDelay = args.eval(0, scope).fusionInt()
11         val initialDelayMillis = args.eval(2, scope).fusionInt() ?: 0L
12         val maxStepsCount = args.eval(3, scope).fusionInt() ?: Long.MAX_VALUE
13         coroutineScope.launch {
14             delay(initialDelayMillis)
15             var currentStep = 0
16             while (coroutineScope.isActive && currentStep < maxStepsCount) {
17                 currentStep++
18                 args.eval(1, scope)
19                 context.reRenderIfNeeded()
20                 delay(stepDelay)
21             }
22         }
23     }
24 }
25
26 return null
27
28 }
29 }
```

Таймер



Lottie



BX-92600

iOS

BX-92601

Android

Провести исследование библиотеки lottie анимации на Android

BX-92674

Fusion

Написать в md файл описание lottie анимации

BX-93046

Fusion

Реализовать dsl атома lottie

BX-93143

Fusion Dsl

Реализовать общий runtime слой для Lottie атома

BX-93144

Fusion Android

Реализовать общий слой Android для Lottie атома

BX-93145

Fusion iOS

Реализовать общий слой iOS для Lottie атома

BX-93208

Fusion Dsl

Покрыть документацией DSL код

BX-93791

Fusion

Реализовать snapshot тесты

BX-94063

iOS

Подключить lottie во fusion в основное приложение

BX-94192

Android

BX-94194

iOS

Поддерживать работу запуска анимации с разных точек

BX-94247

Fusion

Неправильные id у LottieStructure

BX-94274

Fusion

DSL часть запуска анимации с разных точек

BX-94595

Dsl

BX-94594

iOS

BX-94193

Android

Доработать установку текста в лотти анимации

BX-95543

Fusion

Встроить для примера анимацию с lottie в карточку

BX-95353

iOS

BX-95544

Android

Подключить lottie в 1.69 бамп ветке

BX-95572

Lottie

Добавить анимацию в виджет доставки

BX-95761

Fusion iOS

Исправить отображение первого кадра

BX-96055

Fusion

Составить документацию с примером создания молекулы с lottie анимацией

BX-96137

Fusion iOS

Не включается play после stop

BX-96162

Fusion

Необходимо проверить поведение lottie в сложных списках

BX-96225

Fusion

BX-96228

Fusion iOS

Включить лотти для приложений начиная с 8.46.1

```
01 val state = LottieAnimationState(context)
02
03 LottieAnimation(state) {
04     width { fill }
05     height { 200.np }
06     source { "https://st.aestatic.net/resources/test/lottie/test-01.json".c }
07     backgroundColor { pauseAndRestore }
08     resizeMode {
09         aspectFit // (scaleToFill / aspectFill)
10     }
11     isPlaying { true.c }
12     animationSpeed { 1.0.c }
13
14     loopMode {
15         direction {
16             forward // (backward / autoReverse)
17         }
18
19         repeat { 5.c }
20     }
21     onTap { state.reset() }
22 }
```

Lottie Animation

Lottie Animation

```
01 LottieAnimationNode(  
02     viewAttributes = viewAttributes(),  
03     layoutAttributes = layoutAttributes(),  
04     tapAttributes = tapAttributes(),  
05     source = attrOrUndefined(attrs, source) { fusionStringValue() },  
06     placeholder = attrOrUndefined(attrs, placeholder, LottieAnimationNode.Placeholder::parse),  
07     placeholderTintColor = attrOrUndefined(attrs, placeholderTintColor) { parseColorAttr(this) },  
08     errorPlaceholder = attrOrUndefined(attrs, errorPlaceholder, LottieAnimationNode.Placeholder::parse),  
09     errorPlaceholderTintColor = attrOrUndefined(attrs, errorPlaceholderTintColor) { parseColorAttr(this) },  
10     resizeMode = attrOrUndefined(attrs, resizeMode, LottieAnimationNode.ResizeMode::parse),  
11     animationSpeed = attrOrUndefined(attrs, animationSpeed) { fusionFloat()?.toFloat() },  
12     backgroundColor = attrOrUndefined(attrs, backgroundColor, LottieAnimationNode.BackgroundColor::parse),  
13     onAnimationStateChanged = onAnimationStateChanged(),  
14     repeat = attrOrUndefined(attrs, repeat, LottieAnimationNode.Repeat::parse),  
15     direction = attrOrUndefined(attrs, direction, LottieAnimationNode.Direction::parse),  
16     isPlaying = attrOrUndefined(attrs, isPlaying) { fusionBooleanValue() },  
17     min = attrOrUndefined(attrs, min, LottieAnimationNode.Boundary::parse),  
18     max = attrOrUndefined(attrs, max, LottieAnimationNode.Boundary::parse),  
19     textReplacer = attrOrUndefined(attrs, textReplacer) { parseTextReplacer(this) },  
20 )
```

Lottie Animation

```
01 internal object LottieAnimation : ViewRendering<LottieAnimationView, LottieAnimationNode>(),
02     FusionStateHolder<LottieAnimationView, LottieAnimationNode, LottieAnimationAtomState> {
03
04     override fun makeView(fusionView: FusionView, node: LottieAnimationNode): LottieAnimationView =
05         LottieAnimationView(fusionView.context).apply {
06
07     val fontAssetDelegate = LottieFontAssetDelegate(this.context)
08     this.setFontAssetDelegate(fontAssetDelegate)
09
10         // Needed to work for fontAssetDelegate if textReplacer isn't set
11
12         val emptyTextDelegate = LottieTextDelegate(this, mapOf())
13         this.setTextDelegate(emptyTextDelegate)
14     }
```



```
01 internal class LottieFontAssetDelegate(val context: Context) : FontAssetDelegate() {
02     override fun fetchFont(fontFamily: String, fontStyle: String, fontName: String): Typeface {
03         val font = fontName.toFontStyle()
04         return font.typeface(context)
05     }
06
07     private fun String.toFontStyle(): FontStyle {
08         return when (this) {
09             "Inter-Regular" -> FontStyle.Regular
10             "Inter-Bold" -> FontStyle.Bold
11             "Inter-BoldItalic" -> FontStyle.BoldItalic
12             "Inter-ExtraBoldItalic" -> FontStyle.ExtraBoldItalic
13             "Inter-SemiBold" -> FontStyle.SemiBold
14             "Inter-Medium" -> FontStyle.Medium
15             "Ali-Bold" -> FontStyle.AliBold
16             "Ali-Regular" -> FontStyle.AliRegular
17             "Ali-Light" -> FontStyle.AliLight
18             else -> FontStyle.Regular
19         }
20     }
21 }
```

Font Asset Delegate

Resume

```
01 func pause() {  
02     isPaused = true  
03     animationView?.pause()  
04 }  
05  
06 func resume() {  
07     isPaused = false  
08     play(fromProgress: minProgress, to: maxProgress)  
09 }
```

```

01 private fun LottieAnimationView.setRepeat(
02     repeatValue: Int,
03     repeatIsChanged: Boolean,
04     directionValue: LottieAnimationNode.Direction,
05     directionIsChanged: Boolean
06 ) {
07     // it's important to call cancelAnimation() only once when it needed
08     if (repeatIsChanged && !restartNeeded) {
09         cancelAnimation()
10         progress = 0f
11         restartNeeded = true
12     }
13
14     if (repeatIsChanged || directionIsChanged) {
15         if (repeatValue == 0) return
16         repeatCount = when (directionValue) {
17             LottieAnimationNode.Direction.Forward, LottieAnimationNode.Direction.Backward -> {
18                 if (repeatValue == LottieDrawable.INFINITE) LottieDrawable.INFINITE else repeatValue - 1
19             }
20             LottieAnimationNode.Direction.AutoReverse -> {
21                 if (repeatValue == LottieDrawable.INFINITE) LottieDrawable.INFINITE else 2 * repeatValue - 1
22             }
23         }
24     }
25 }
26 }

```

Auto Reverse

LottieAnimationNode.Direction.AutoReverse

Reverse

01 When the animation reaches the end and `repeatCount` is INFINITE
02 or a positive value, the animation restarts from the beginning.

03
04 `public static final int RESTART = 1;`

05
06 When the animation reaches the end and `repeatCount` is INFINITE
07 or a positive value, the animation reverses direction on every iteration.

08
09 `public static final int REVERSE = 2;`

10
11 This value used used with the `setRepeatCount(int)` property to repeat
12 the animation indefinitely.

13
14 `public static final int INFINITE = -1;`

```

01 private fun LottieAnimationView.setSpeedAndDirection(
02     animationSpeed: FusionAttribute<Float?>,
03     directionValue: LottieAnimationNode.Direction,
04     directionIsChanged: Boolean,
05 ) {
06     val speedValue = animationSpeed.value ?: 1f
07     if (animationSpeed.isChanged() || directionIsChanged) {
08         when (directionValue) {
09             LottieAnimationNode.Direction.Forward -> {
10                 repeatMode = LottieDrawable.RESTART
11                 speed = speedValue
12             }
13
14             LottieAnimationNode.Direction.Backward -> {
15                 repeatMode = LottieDrawable.RESTART
16                 speed = -speedValue
17             }
18
19             LottieAnimationNode.Direction.AutoReverse -> {
20                 repeatMode = LottieDrawable.REVERSE
21                 speed = speedValue
22             }
23         }
24     }
25 }

```

Auto Reverse

```

01 private fun LottieAnimationView.setRepeat(
02     repeatValue: Int,
03     repeatIsChanged: Boolean,
04     directionValue: LottieAnimationNode.Direction,
05     directionIsChanged: Boolean
06 ) {
07     // it's important to call cancelAnimation() only once when it needed
08
09     if (repeatIsChanged && !restartNeeded) {
10         cancelAnimation()
11         progress = 0f
12         restartNeeded = true
13     }
14
15     if (repeatIsChanged || directionIsChanged) {
16         if (repeatValue == 0) return
17         repeatCount = when (directionValue) {
18             LottieAnimationNode.Direction.Forward, LottieAnimationNode.Direction.Backward -> {
19                 if (repeatValue == LottieDrawable.INFINITE) LottieDrawable.INFINITE else repeatValue - 1
20             }
21             LottieAnimationNode.Direction.AutoReverse -> {
22                 if (repeatValue == LottieDrawable.INFINITE) LottieDrawable.INFINITE else 2 * repeatValue - 1
23             }
24         }
25     }
26 }
27 }

```

Repeat Value

repeatValue - 1

```

01 private fun LottieAnimationView.setAnimationParams(
02     isPlaying: FusionAttribute<Boolean?>,
03     animationSpeed: FusionAttribute<Float?>,
04     direction: FusionAttribute<LottieAnimationNode.Direction?>,
05     repeat: FusionAttribute<LottieAnimationNode.Repeat?>,
06     min: FusionAttribute<LottieAnimationNode.Boundary?>,
07     max: FusionAttribute<LottieAnimationNode.Boundary?>,
08 ) {
09     val isPlayingValue = isPlaying.value ?: false
10     val directionValue = direction.value ?: LottieAnimationNode.Direction.Forward
11     val directionIsChanged = direction.isChanged()
12     val repeatValue = repeat.value.toInt()
13     setRepeat(repeatValue, repeat.isChanged(), directionValue, directionIsChanged)
14     setMinBoundary(min)
15     setMaxBoundary(max)
16     setSpeedAndDirection(animationSpeed, directionValue, directionIsChanged)
17
18     if (isPlaying.isChanged()) {
19         if (isPlayingValue) {
20             if (!animationEnded && repeatValue != 0) {
21                 if (animationPaused) resumeAnimation() else playAnimation()
22                 animationPaused = false
23             }
24         } else {
25             if (!restartNeeded) {
26                 if (isAnimating) animationPaused = true
27                 pauseAnimation()
28             }
29         }
30     } else {
31         if (isPlayingValue && restartNeeded && repeatValue != 0) playAnimation()
32     }
33
34     restartNeeded = false
35 }
36

```

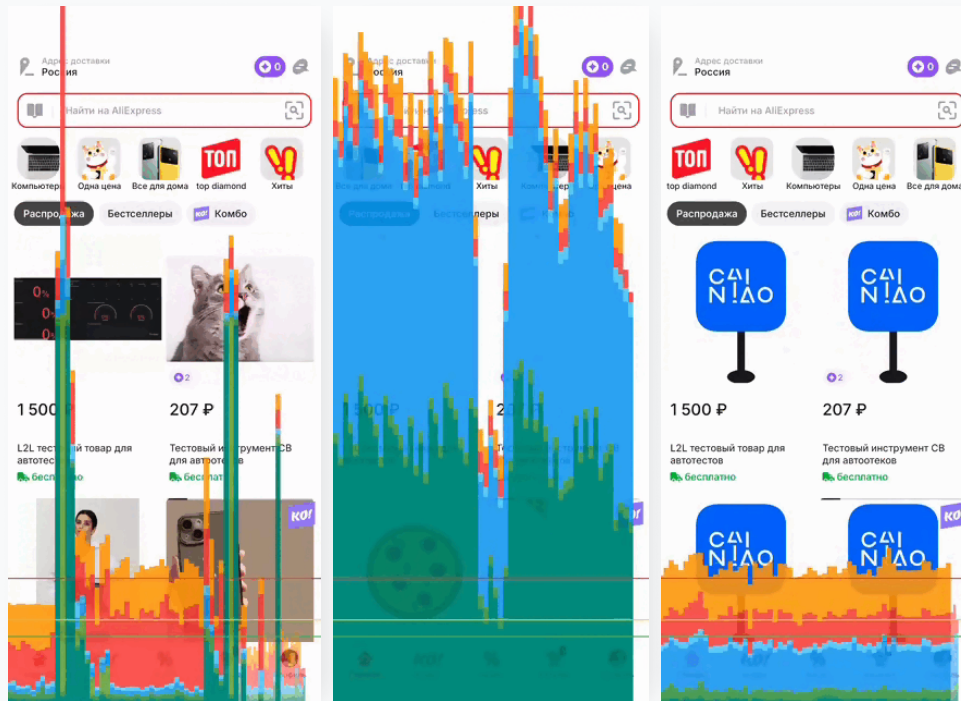
Restart Needed

```

01 private fun LottieAnimationView.setRepeat(
02     repeatValue: Int,
03     repeatIsChanged: Boolean,
04     directionValue: LottieAnimationNode.Direction,
05     directionIsChanged: Boolean
06 ) {
07     // it's important to call cancelAnimation() only once when it needed
08     if (repeatIsChanged && !restartNeeded) {
09         cancelAnimation()
10         progress = 0f
11         restartNeeded = true
12     }
13
14     repeatCount = when (directionValue) {
15         LottieAnimationNode.Direction.Forward, LottieAnimationNode.Direction.Backward -> {
16             if (repeatValue == LottieDrawable.INFINITE) LottieDrawable.INFINITE else repeatValue - 1
17         }
18         LottieAnimationNode.Direction.AutoReverse -> {
19             if (repeatValue == LottieDrawable.INFINITE) LottieDrawable.INFINITE else 2 * repeatValue - 1
20         }
21     }
22 }

```

Restart Needed



Android

мб

Без анимации

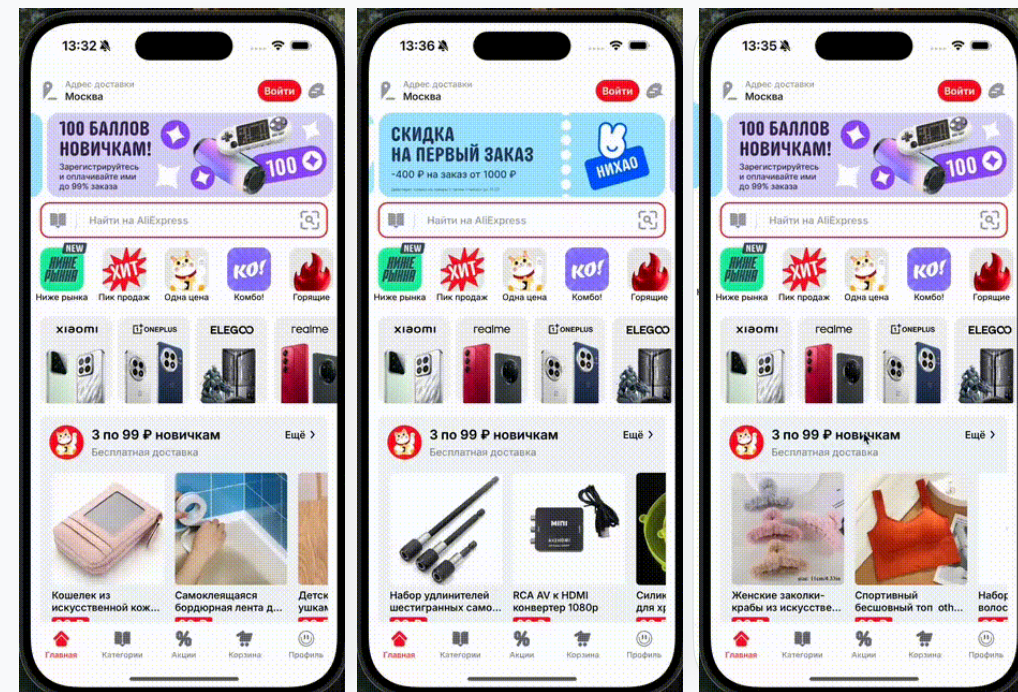
861,8

Анимация Цайняо

984,1

Анимация Пицца

822,8



iOS

мб

Без анимации

614,8

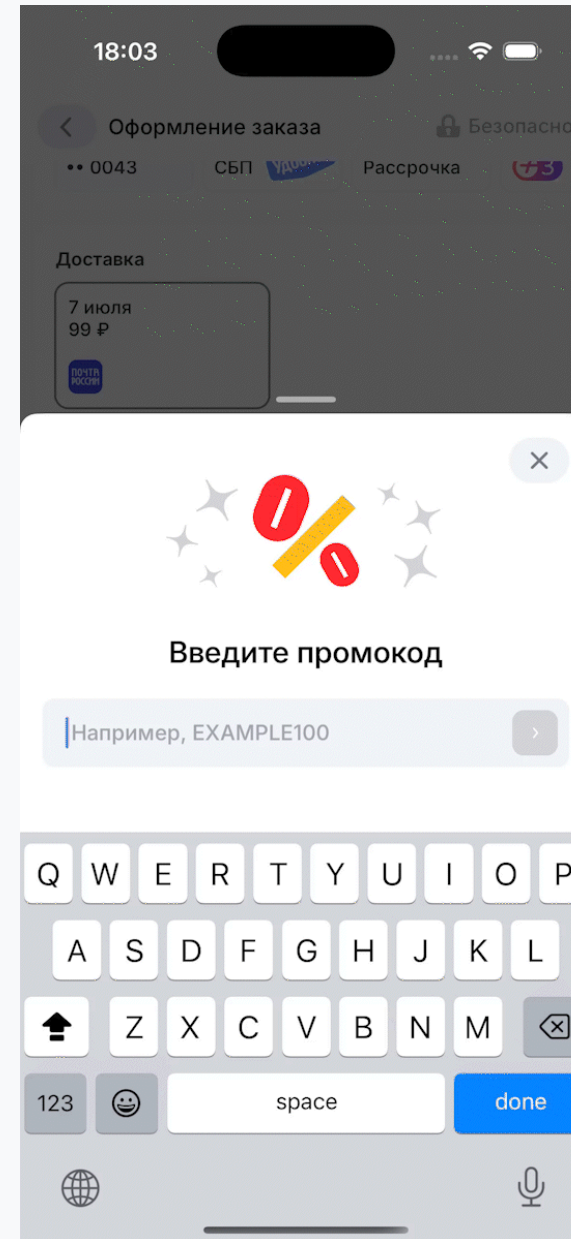
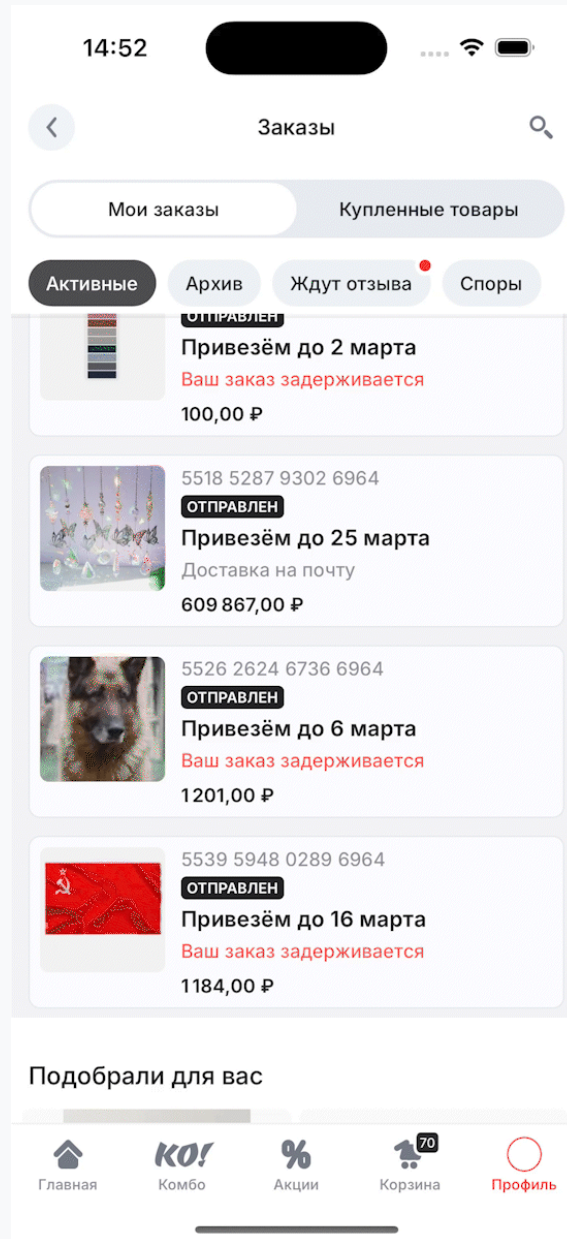
Анимация Цайняо

611,4

Анимация Пицца

642,7

Анимация на проде



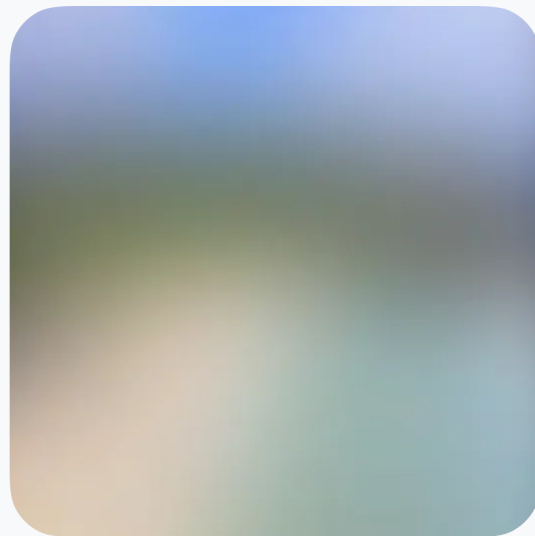
эффект размытия ios



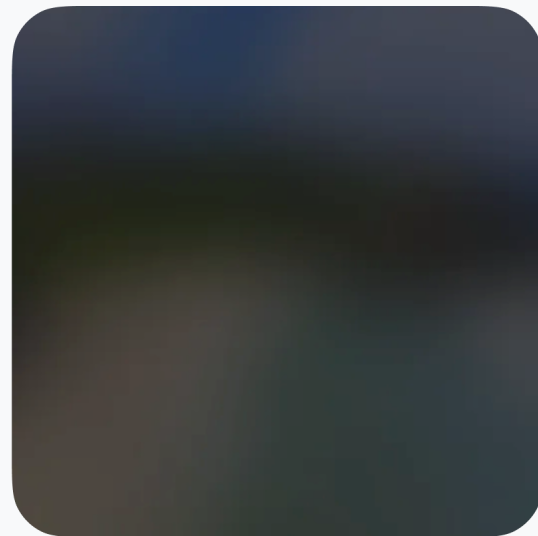
none



extraLight



light



dark

эффект размытия android



none



render effect

r50



compose blur

r50



Coil

+

BlurTransformation

r50

Android

Render Effect

Размывает View
и её содержимое

Гибкие настройки

iOS

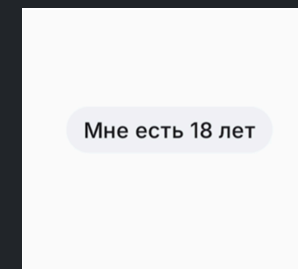
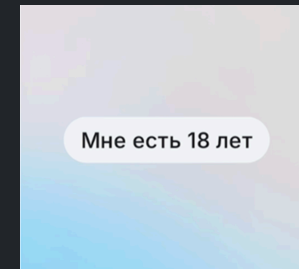
UI Blur Effect

Создаёт прозрачную View,
которая размывает всё,
что под ней

3 стиля на выбор

Blur пример

```
01 // Примерный код внутри молекулы
02 Box {
03     // Используя функцию isBlurSupported() мы отображаем либо выюху, которая будет блюриться в случае необходимости -> viewModel.isViewShouldBlur(node)
04     // Либо отображаем аналогичную выюху, которая не умеет блюриться и мы в ней сами (вместе с дизайнерами) придумываем, как обработать этот кейс
05
06     // Этот бокс будет отображаться, если isBlurSupported() и будет блюриться если viewModel.isViewShouldBlur(node)
07     Box {
08         isBlurred { viewModel.isViewShouldBlur(node) }
09         isVisible { isBlurSupported() }
10         // Здесь контент, который возможно нужно блюрить
11     }
12
13     // Этот бокс будет отображаться если НЕ isBlurSupported() и будет закрывать контент картинкой если viewModel.isViewShouldBlur(node)
14     Box {
15         isVisible { isBlurSupported().not() }
16         // Здесь контент, который возможно нужно скрыть от детишек
17
18         Image {
19             width { fill }
20             height { fill }
21             // Это большой серый костыль закрывающий контент или что-то другое вместо блюра
22             isVisible { viewModel.isViewShouldBlur(node) }
23         }
24     }
25 }
```



```
01 internal object IsBlurSupported : FusionFunction {
02     override val id: FunctionId = FunctionIds.isBlurSupported
03
04     override val isPure: Boolean = true
05
06     override fun eval(
07         args: FusionFunction.Args,
08         context: FusionContext,
09         scope: FusionScope?
10     ): Boolean {
11         return isBlurSupported()
12     }
13 }
14
15 /**
16  * Returns true if blur supported on platform
17  */
18 internal expect fun isBlurSupported(): Boolean
```

Fusion
Is Blur Supported

KMM Is Blur Supported

```
01 /**
02  * Returns true if blur supported on ios
03  */
04 actual fun isBlurSupported(): Boolean = true
05
06 /**
07  * Returns true if blur supported on android
08  */
09 actual fun isBlurSupported(): Boolean {
10     return (getAndroidSdkVersion() ?: 0) >= Build.VERSION_CODES.S
11 }
```

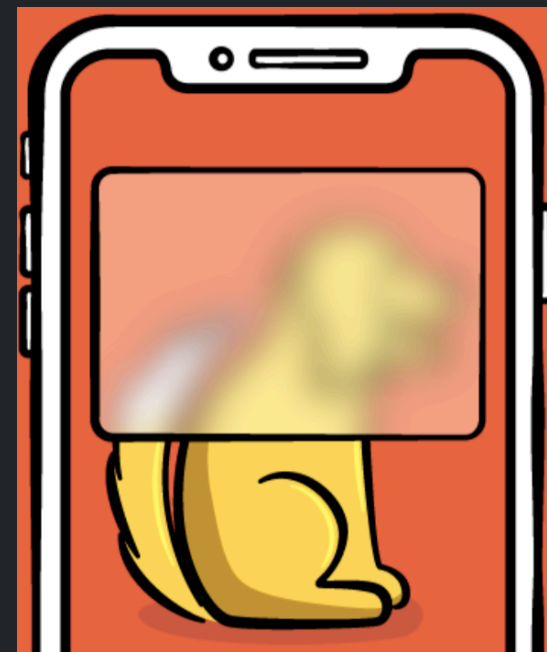

Blur iOS

```
01 extension UIViewEffectView {  
02     fileprivate static func makeBlurringView() -> UIViewEffectView {  
03         let effect = UIViewEffectView(effect: UIBlurEffect(style: .systemThinMaterial))  
04         return effect  
05     }  
06 }
```



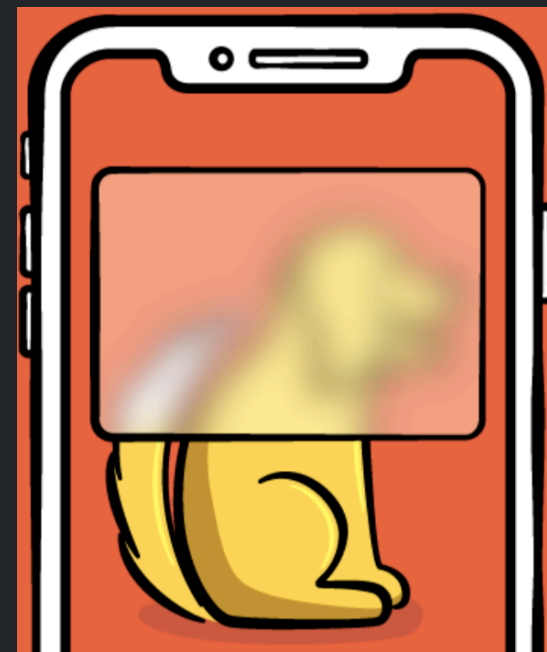
Blur iOS

```
01 private func configureBlur() {  
02     if node.isBlurred {  
03         if let blurringView {  
04             addSubview(blurringView)  
05         } else {  
06             let blurringView = UIVisualEffectView.makeBlurringView()  
07             self.blurringView = blurringView  
08             addSubview(blurringView)  
09         }  
10     } else {  
11         blurringView?.removeFromSuperview()  
12     }  
13 }
```



Blur iOS

```
01 /// Provides common logic for subclasses.
02 func nodeUpdated(updateChildrenViewsClosure: UpdateChildrenViewsClosure)
03     performUpdates {
04         blurringView?.removeFromSuperview()
05         configureChildrenViews(updateChildrenViewsClosure)
06         configureBlur()
07     }
08 }
```



```
01 /**
02  * Designed for container views. Blurs content. Works starting with Android 12
03  * @param isEnabled turns the effect on or off
04  */
05 @RequiresApi(Build.VERSION_CODES.S)
06 fun View.blurEffect(isEnable: Boolean) {
07     val effect = if (isEnabled) {
08         val radius = 180f
09         setTag(CLIP_TO_OUTLINE_TAG, clipToOutline)
10         this.clipToOutline = true
11         RenderEffect.createBlurEffect(
12             radius,
13             radius,
14             Shader.TileMode.CLAMP,
15         )
16     } else {
17         (getTag(CLIP_TO_OUTLINE_TAG) as? Boolean)?.let { this.clipToOutline = it }
18         null
19     }
20     this.setRenderEffect(effect)
21 }
```

Blur Android

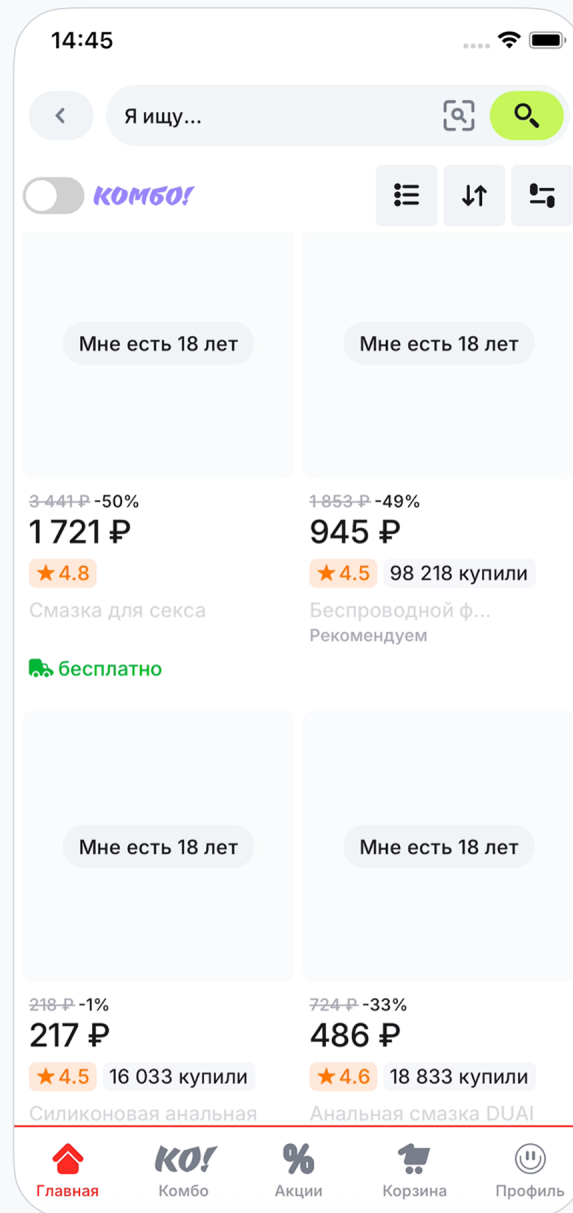
Blur Android

```
01 private fun View.renderBlurIfSupported(node: LayoutNode) {  
02     if (Build.VERSION.SDK_INT < Build.VERSION_CODES.S) return  
03     if (!node.children.isBlurred.isChanged()) return  
04     blurEffect(node.children.isBlurred.value == true)  
05 }
```

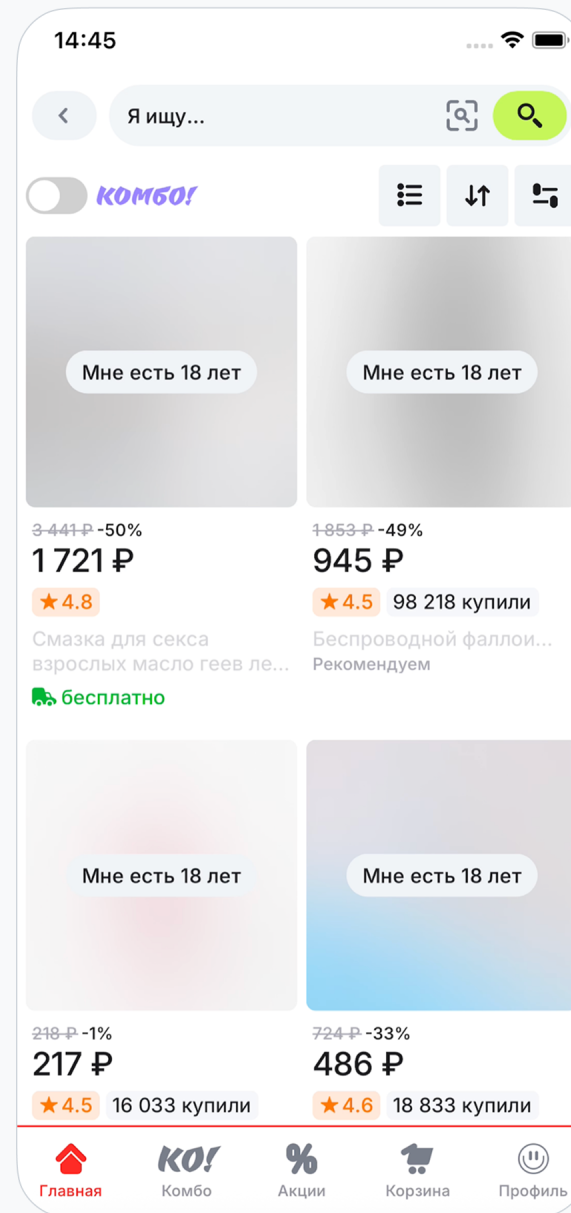
Blur Android

```
01 override fun updateView(  
02     view: TView,  
03     node: TNode,  
04     fusionView: FusionView,  
05 ) {  
06     super.updateView(view, node, fusionView)  
07     view.renderBlurIfSupported(node)  
08  
09  
10     if (!node.children.isChanged()) return  
11  
12     if (node.children.isLoading.value == true) {  
13         updateChildren(view, node.children.loadingItems.value, fusionView, node.atomUniqueId)  
14     } else {  
15         updateChildren(view, node.children.children.value, fusionView, node.atomUniqueId)  
16     }  
17 }
```

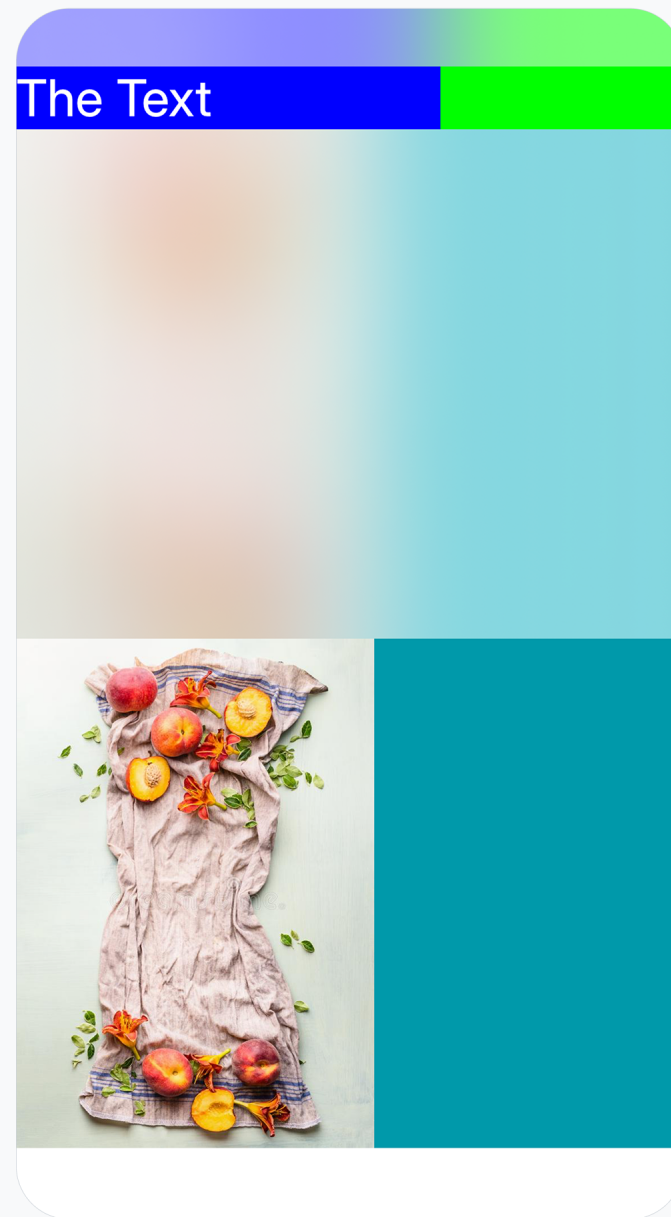
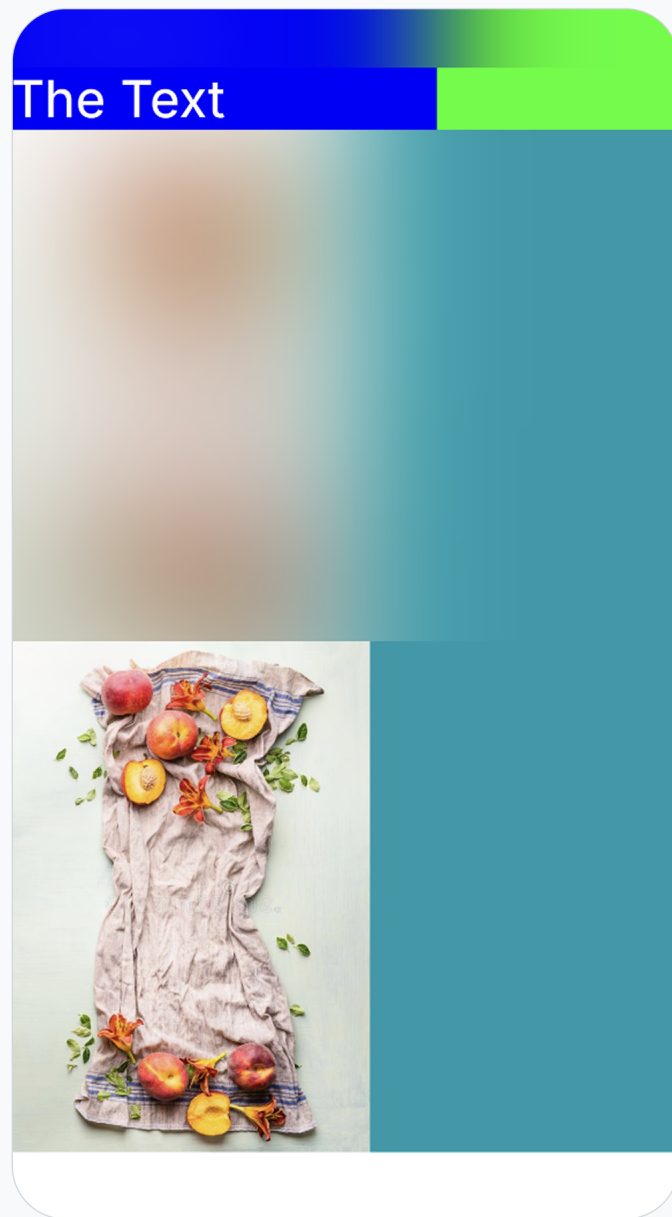
ДО



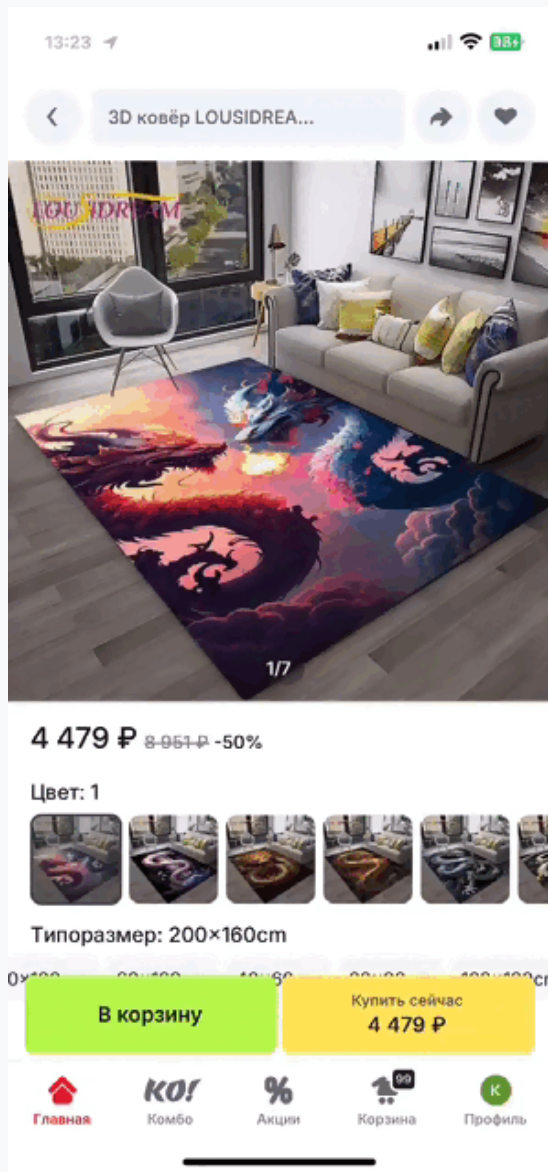
ПОСЛЕ



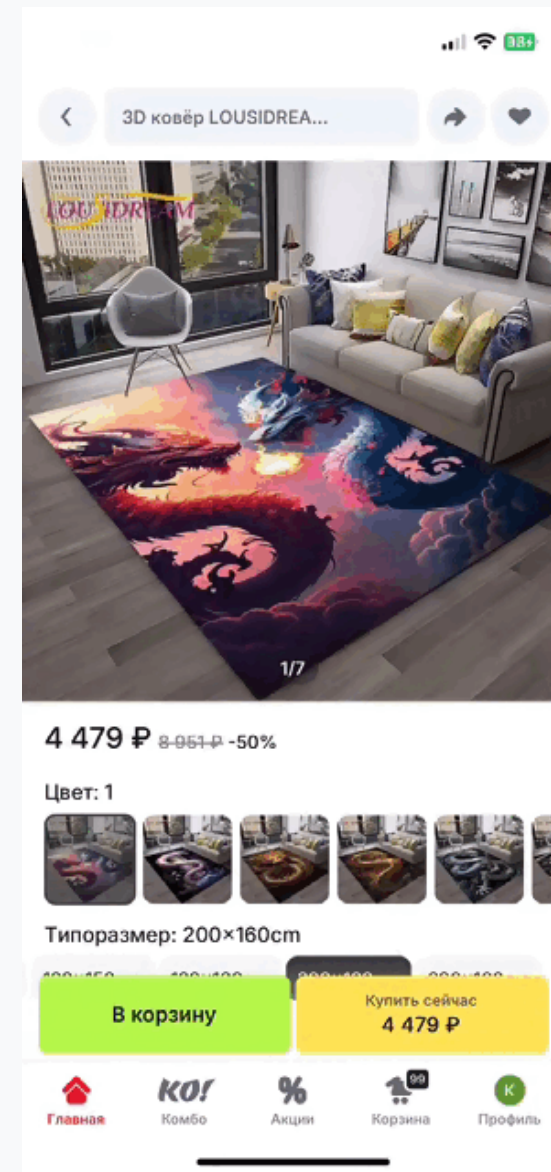
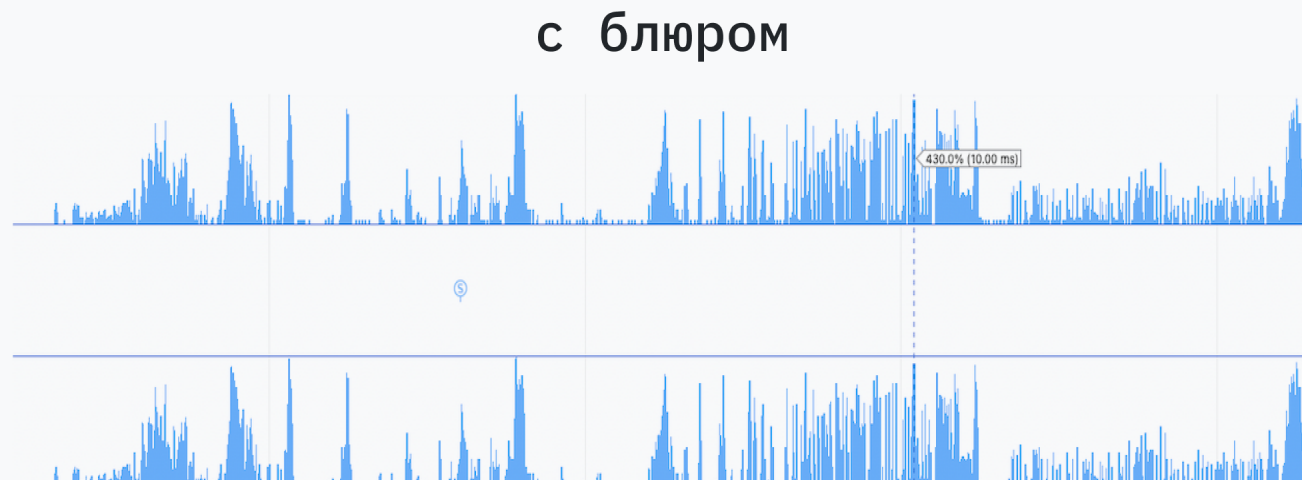
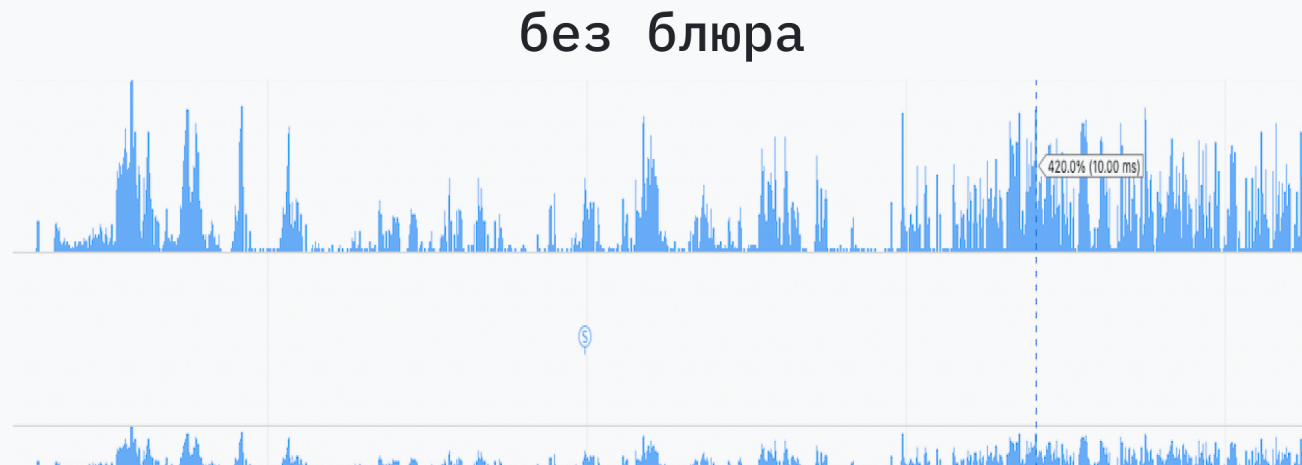
ios



android

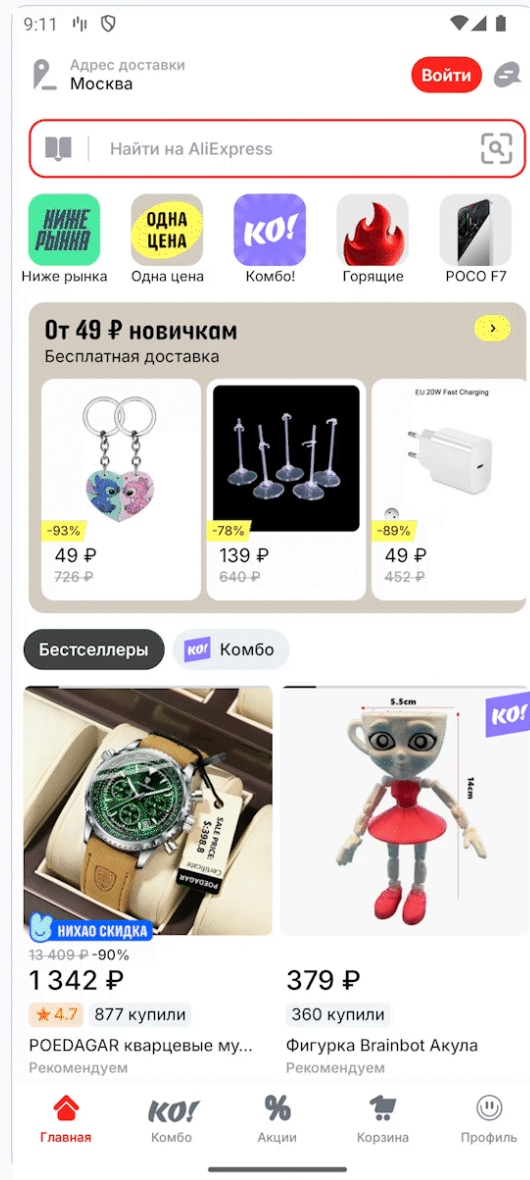
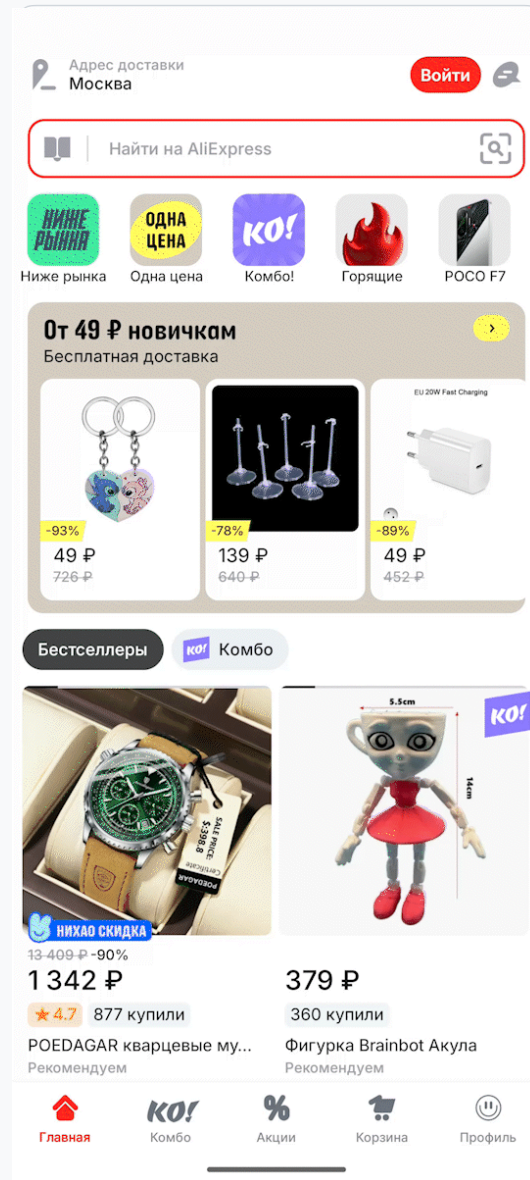


no blur

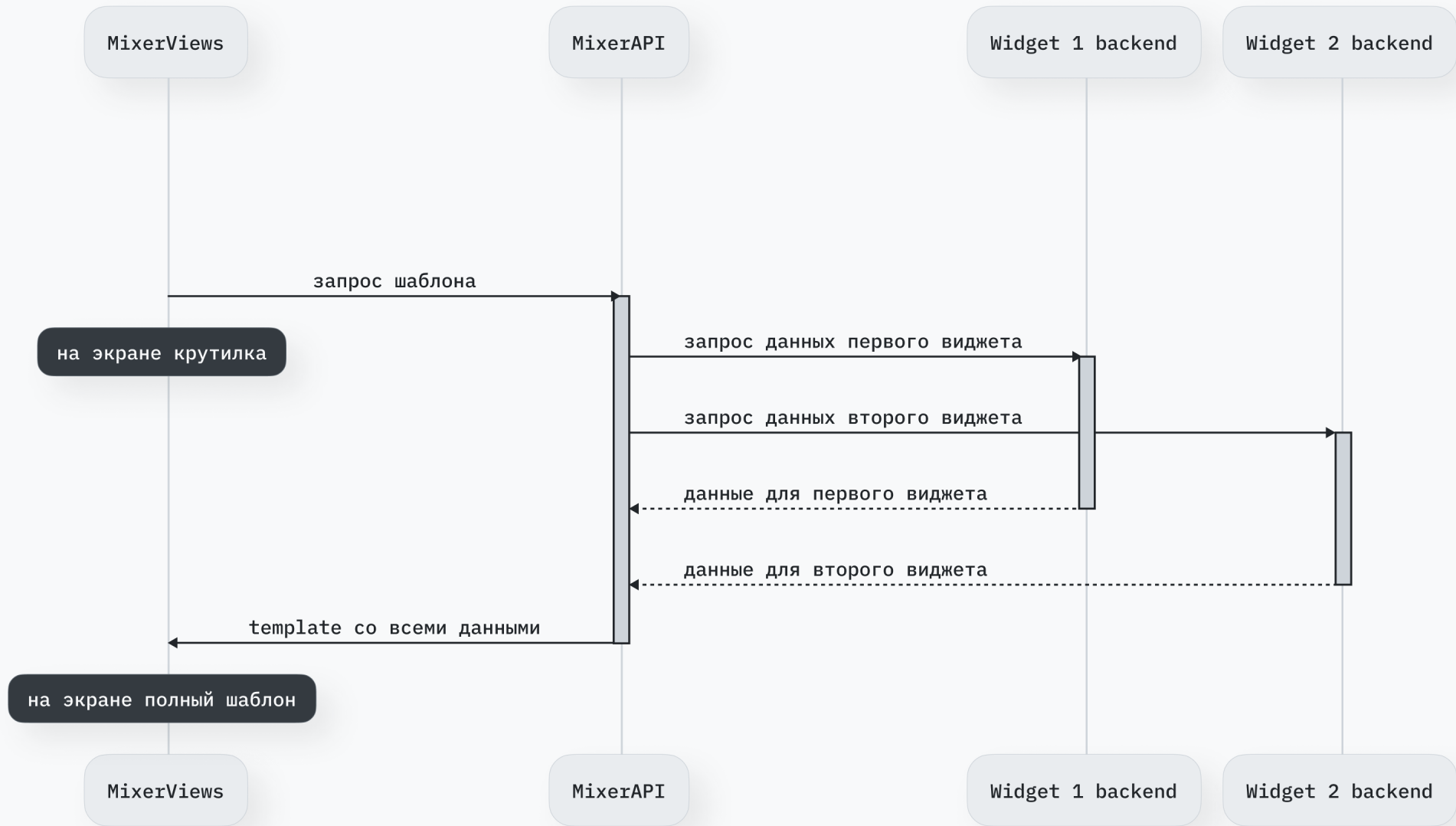


blur

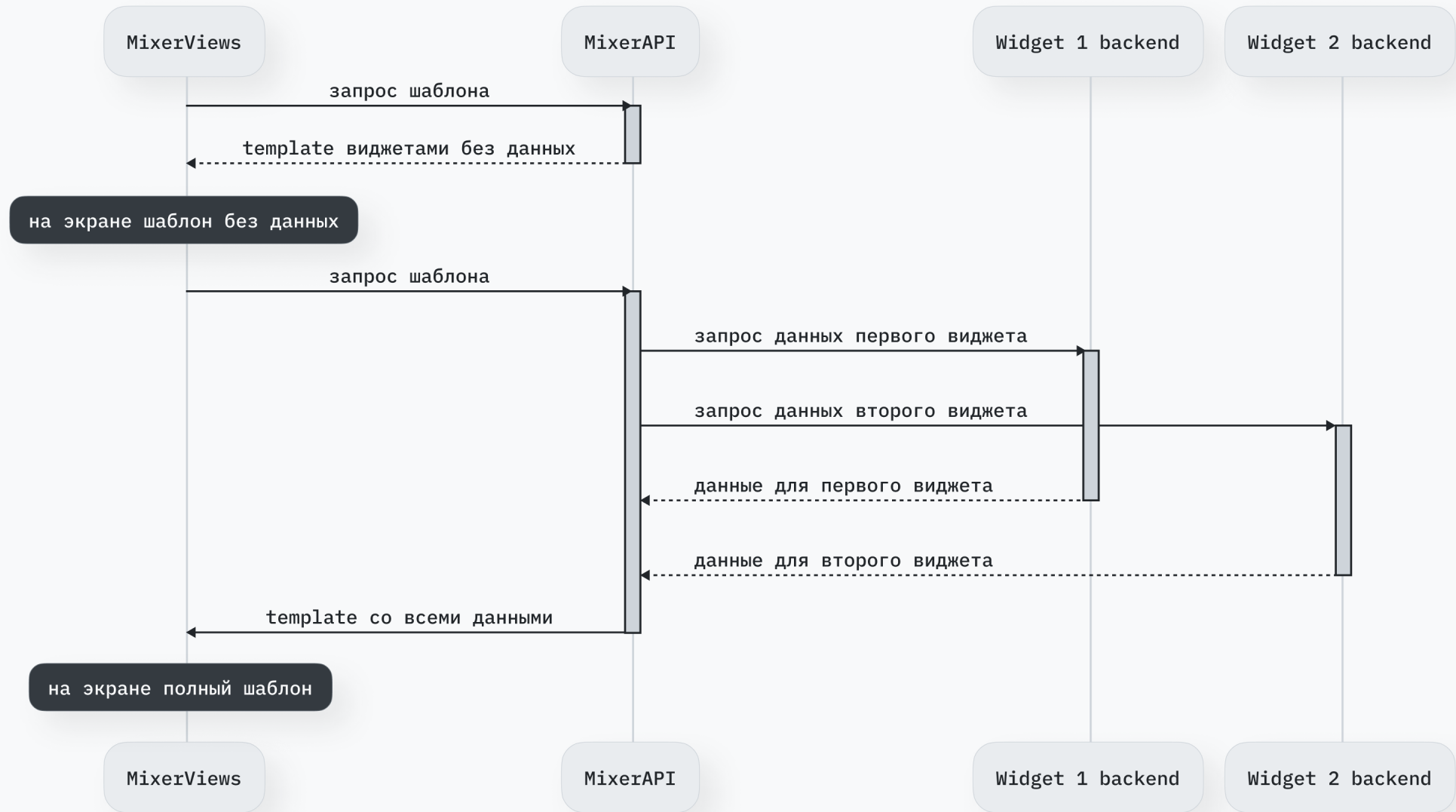
с плавным
переходом



через
лоадер



отображение шаблона с синхронными виджетами (лоадер)



отображение меташаблона с последующей подгрузкой данных (плавный переход)

Мета Шаблон

```
01 val MyScreen by molecule(  
02     Version(...),  
03     includeInMetaTemplate = true,  
04     ...  
05 ) { }
```

Мета Шаблон

```
01 class MyViewModel(...) : BaseViewModel(context) {  
02     ...  
03     init {  
04         ...  
05         onAsyncBackendStateAvailable {  
06             loadMetaTemplate(FEATURE_PATH)  
07         }  
08     }
```



Alexey Telyshev

Какое-то заключение о том, что ЮИ важен для разработчиков, менеджеров и, главное, для пользователя. Что нужно не забывать иногда его обновлять и дорабатывать. Даже если это сложный БДЮИ движок

Телышев Алексей



perfectiist_live

Дизайнер Даниил Ермоленко

Живая
картинка
анимации
и эффекты
в кроссплат-
форменном
BDUI-движке



cherelayka

Вопросы

Телышев Алексей