bumble badoo

Дизайн-система на Jetpack Compose

Антон Шилов

Bumble

Badoo

~1M

Строк кода

~5000

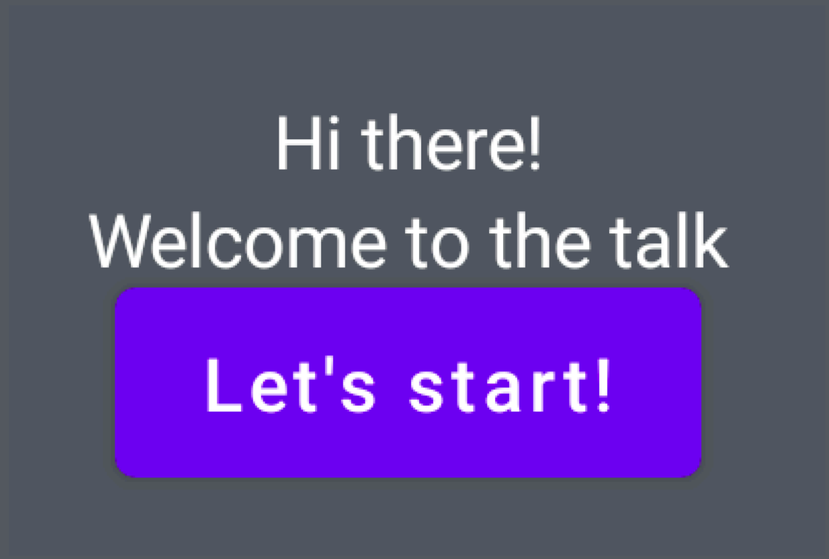
UI-тестов

~100

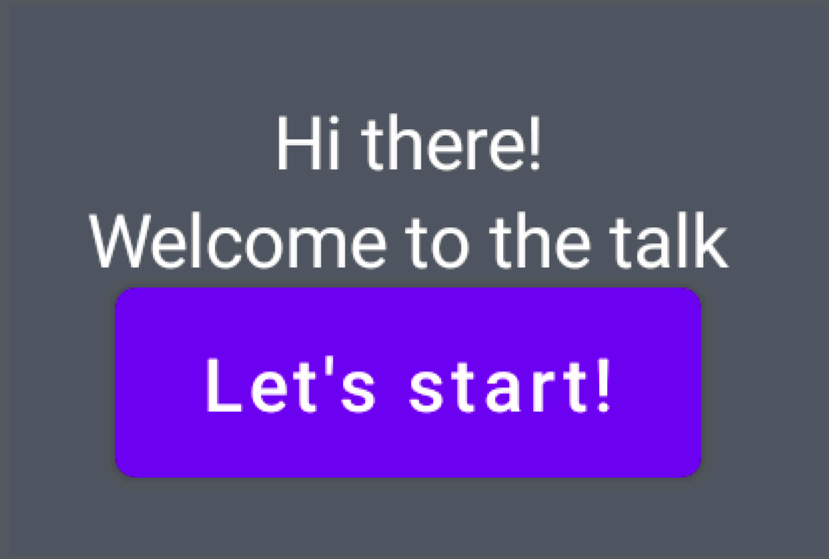
UI-компонентов

Jetpack Compose — декларативный UI фреймворк на Kotlin

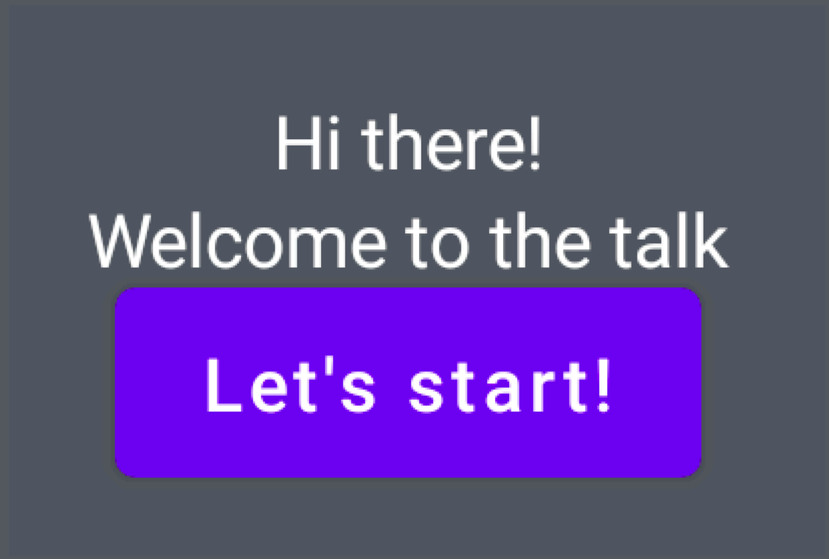
```
@Composable
fun HelloMobius() {
    Column{
        Text(text = "Hi there!")
        Text(text = "Welcome to the talk")
        Button(onClick = {}) {
            Text(text = "Let's start!")
        }
    }
}
```



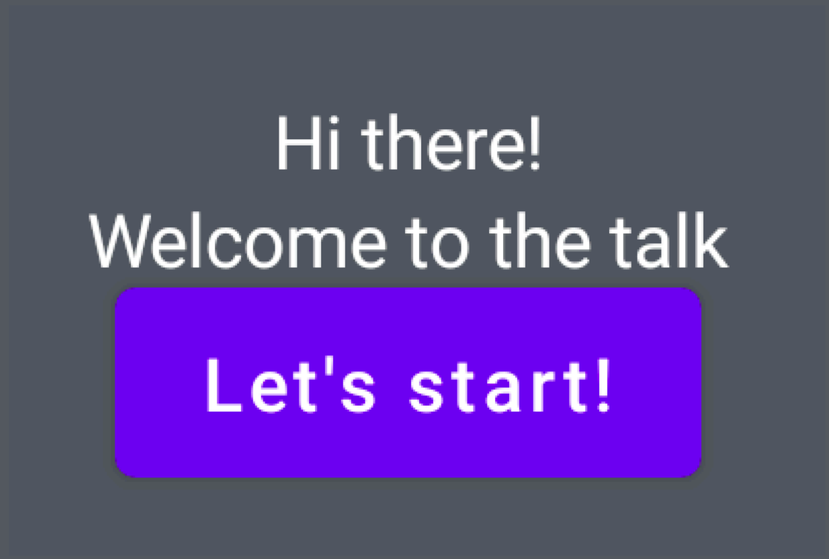
```
@Composable
fun HelloMobius() {
    Column{
        Text(text = "Hi there!")
        Text(text = "Welcome to the talk")
        Button(onClick = {}) {
            Text(text = "Let's start!")
        }
    }
}
```



```
@Composable
fun HelloMobius() {
    Column{
        Text(text = "Hi there!")
        Text(text = "Welcome to the talk")
        Button(onClick = {}) {
            Text(text = "Let's start!")
        }
    }
}
```



```
@Composable
fun HelloMobius() {
    Column{
        Text(text = "Hi there!")
        Text(text = "Welcome to the talk")
        Button(onClick = {}) {
            Text(text = "Let's start!")
        }
    }
}
```



Зачем нам Compose?

- Будущее Android UI
- Ускорение разработки
- Отлично ложится на нашу архитектуру — MVI
- Свой декларативный UI фреймворк
- Легко создавать дизайн-системы

Цель 🎯

Начать использовать Compose после стабильного релиза

Вопросы

- Как реализовать дизайн-систему?
- Как интегрировать с текущими экранами?
- Как тестировать?
- Как интегрировать с логикой?

Дизайн система — набор правил и инструментов для создания UI

Текстовые стили

Текстовые стили

Текстовые стили

Текстовые стили

Дизайн токены

Размеры

Цвета

Иконки

Tottenham Court Road London Underground

Tottenham Court Road, W1D 2DA
Greater London, United Kingdom

Lorem ipsum dolor

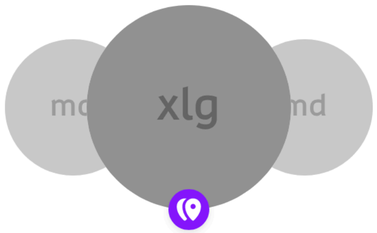
Компоненты

Lorem ipsum

Lorem ipsum dolor sit amet, consectetur.

Lorem ipsum dolor sit amet, consectetur adipisicing elit in officia placeat quasi quisquam quod quos.

Lorem ipsum



Lorem ipsum dolor sit amet, consectetur.

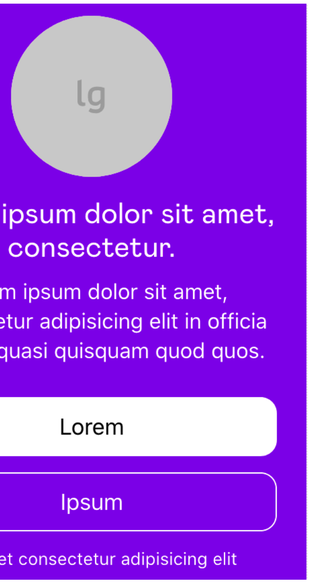
Lorem ipsum dolor sit amet, consectetur adipisicing elit in officia placeat quasi quisquam quod quos.

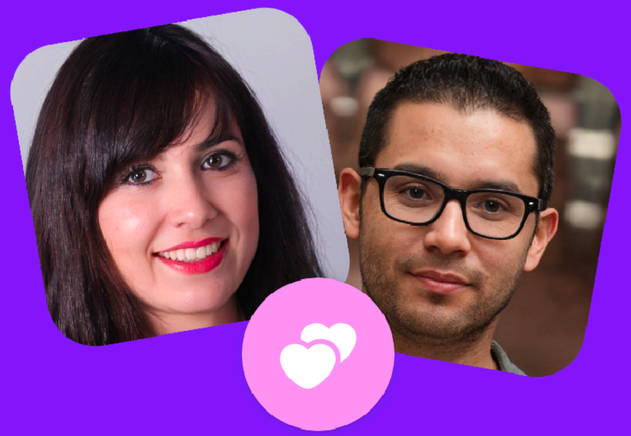
Lorem

Ipsum

Dolor

Паттерны





You have a new match!

Hugo matched with you while you were away. Now’s the perfect time to send them a message

Send Message

You have a new match!

Hugo matched with you while you were away. Now’s the perfect time to send them a message

Send Message

CtaBox

MatchPhotos

ic badge-feature-match

Text - H1

Text - P1

Button - Filled - White

You have a new match!

Hugo matched with you while you were away. Now’s the perfect time to send them a message

Send Message

```
data class TextStyle(
    val color: Color = Color.Unspecified,
    val fontSize: TextUnit = TextUnit.Inherit,
    val fontWeight: FontWeight? = null,
    val fontStyle: FontStyle? = null,
    val fontSynthesis: FontSynthesis? = null,
    val fontFamily: FontFamily? = null,
    val fontFeatureSettings: String? = null,
    val letterSpacing: TextUnit = TextUnit.Inherit,
    val baselineShift: BaselineShift? = null,
    val textGeometricTransform: TextGeometricTransform? = null,
    val localeList: LocaleList? = null,
    val background: Color = Color.Unspecified,
    val textDecoration: TextDecoration? = null,
    val shadow: Shadow? = null,
    val textAlign: TextAlign? = null,
    val textDirection: TextDirection? = null,
    val lineHeight: TextUnit = TextUnit.Inherit,
    val textIndent: TextIndent? = null
)
```

```
H1 = TextStyle(
    fontFamily = FontFamily.Default,
    fontWeight = FontWeight.Medium,
    fontSize = 24.sp,
    lineHeight = 28.sp
)
```

Hello Mobius!

```
data class Typography(
    val H1: TextStyle,
    val H2: TextStyle,
    val Action: TextStyle,
    val Title: TextStyle,
    val P1: TextStyle,
    val P2: TextStyle,
    val P3: TextStyle
)
```

```
colorResource(id = R.color.black)
Color(0xFF783bf9)
```

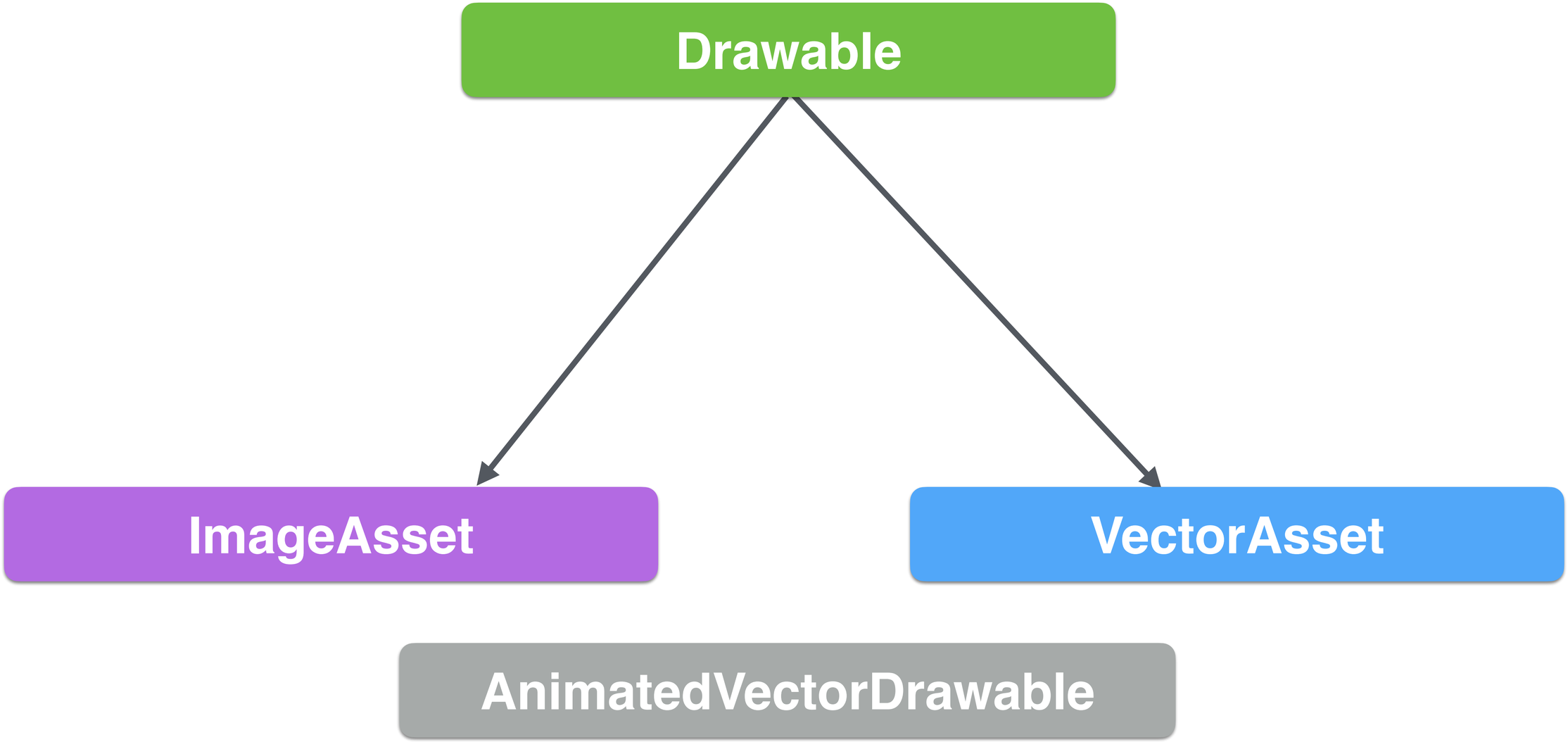
```
data class Palette(
    val primary: Color,
    val matchPhotosBorder: Color,
    // ...
)
```

```
val sizeDp = 16.dp
val sizeSp = 16.sp
dimensionResource(id = R.dimen.size_lg)
```

```
inline val Int.dp: Dp get() = Dp(value = this)
```

```
data class DimensionsTokens(
    val tokenMatchPhotosPhotoSize: Dp,
    val tokenButtonHeight: Dp,
    val tokenButtonIconSize: Dp,
    val tokenButtonIconTextSpacing: Dp,
    val tokenButtonDisabledOpacity: Float,
    val tokenButtonStrokeBorderWidth: Dp,
    val tokenMatchPhotosBorderWidth: Dp,
    // ...
)
```

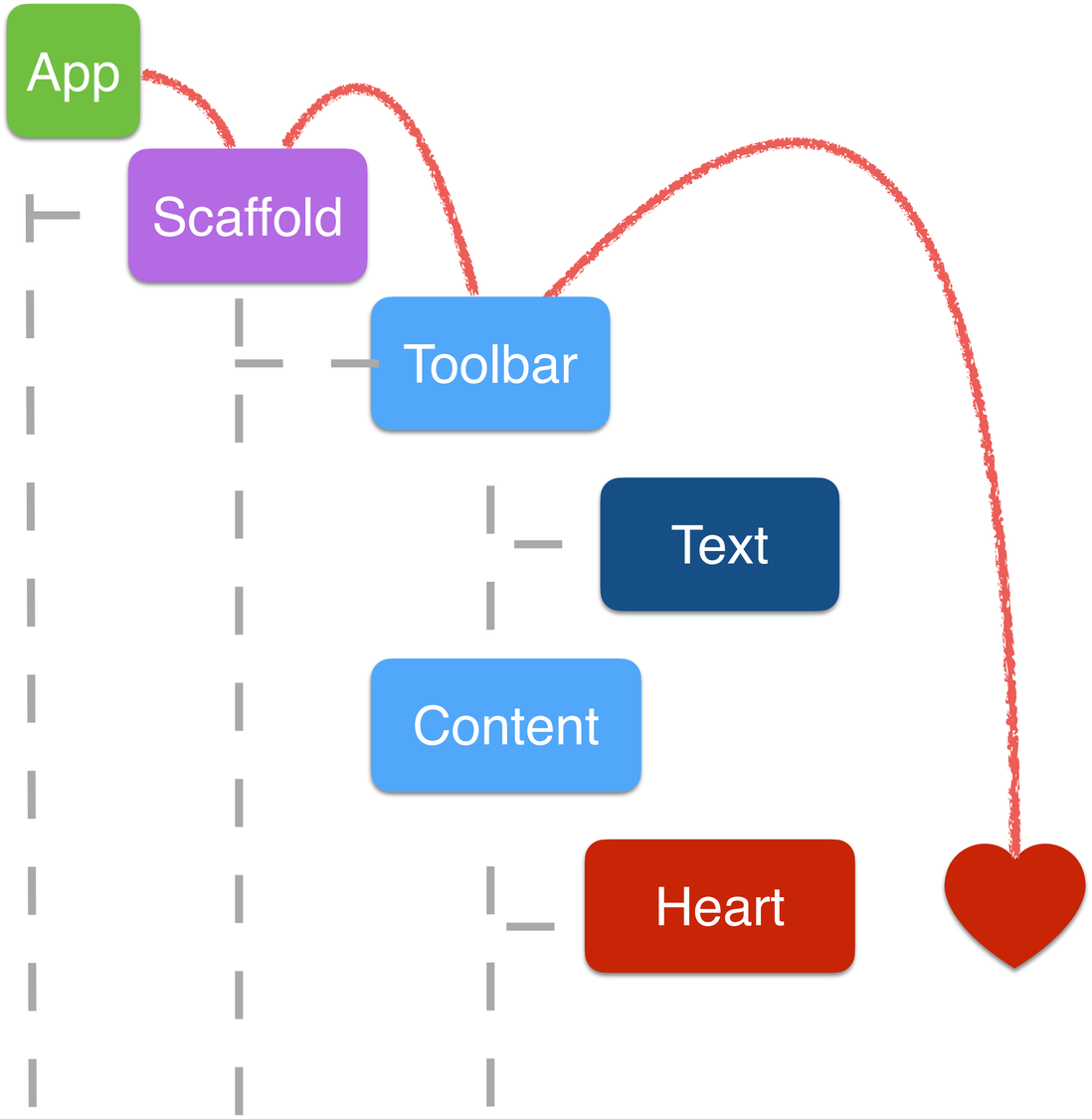
Иконки

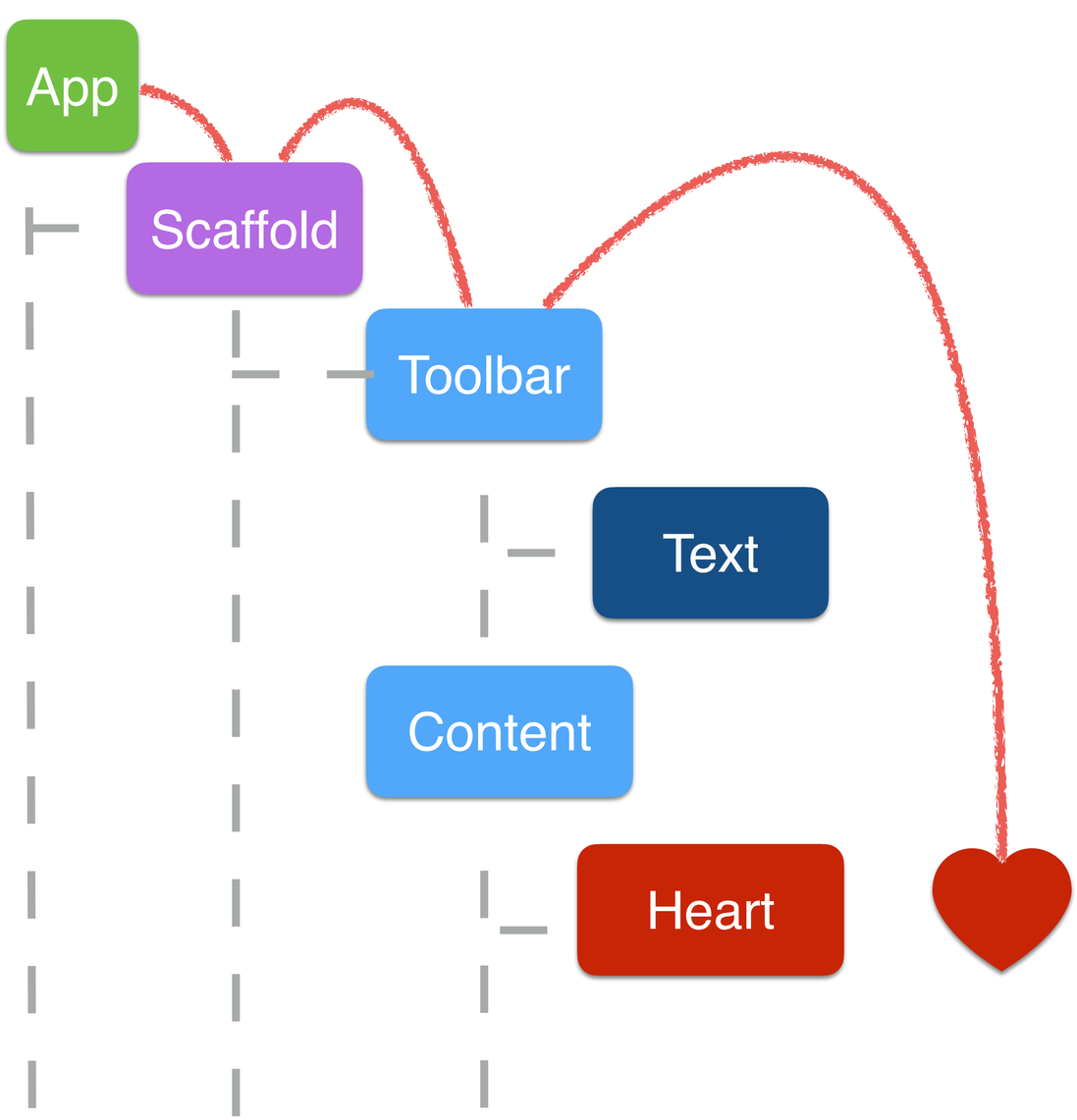

Drawable
ImageAsset
VectorAsset
AnimatedVectorDrawable

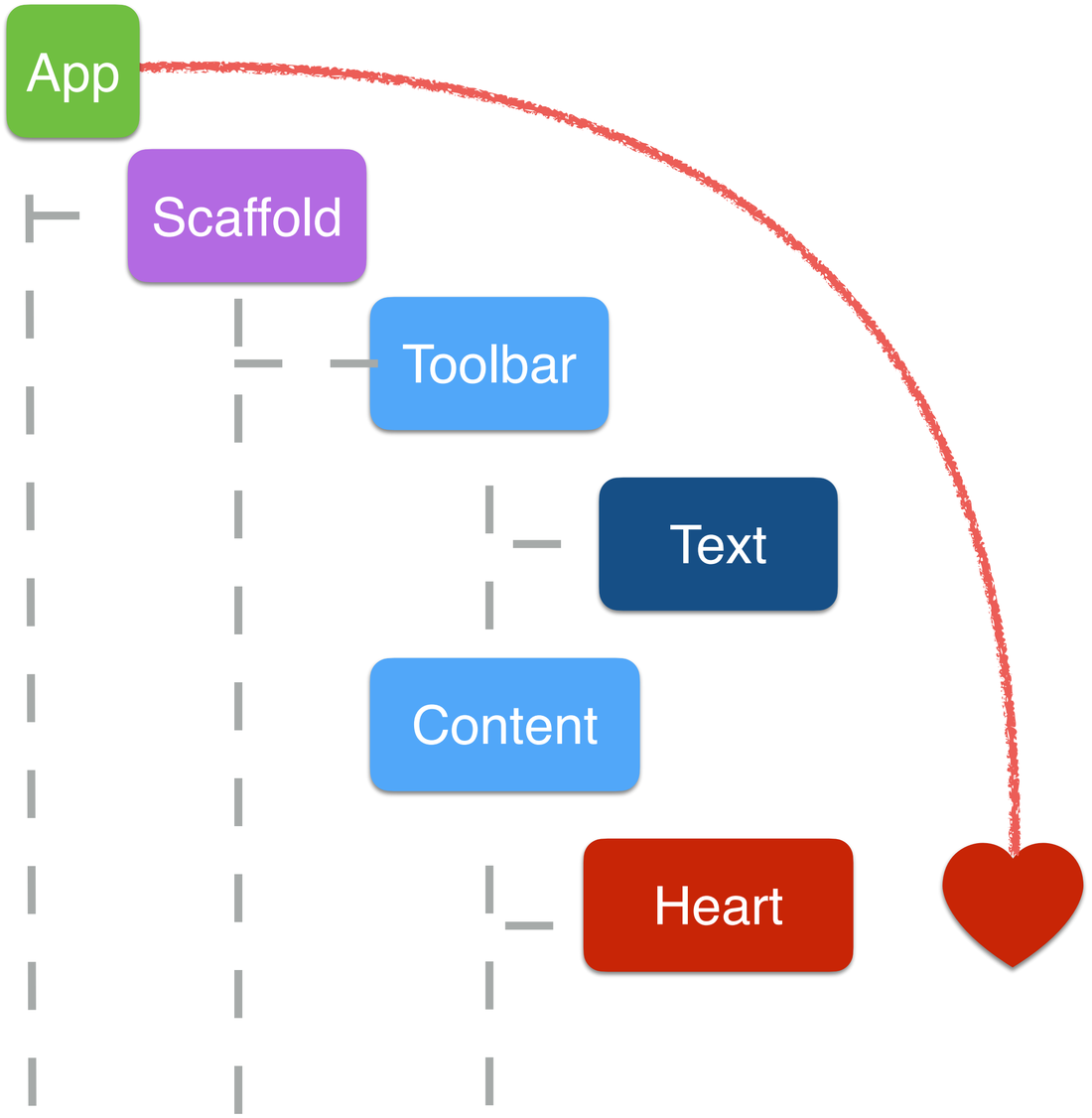
```
imageResource(id = R.drawable.photo)
vectorResource(id = R.drawable.ic_match)
```

```
val Icons.Filled.TestVector: VectorAsset
    get() =
        materialIcon(name = "Filled.TestVector") {
            materialPath(fillAlpha = 0.8f) {
                moveTo(20.0f, 10.0f)
                lineToRelative(0.0f, 10.0f)
                lineToRelative(-10.0f, 0.0f)
                close()
            }
            group {
                materialPath(pathFillType = EvenOdd) {
                    moveTo(0.0f, 10.0f)
                    lineToRelative(-10.0f, 0.0f)
                    close()
                }
            }
        }
```

Стили + Токены = Тема


App
Scaffold
Toolbar
Text
Content
Heart


App
Scaffold
Toolbar
Text
Content
Heart


App
Scaffold
Toolbar
Text
Content
Heart

```
val AmbientHeartColor = ambientOf<Color>()

@Composable
fun HeartsScreen() {
    Providers(
        AmbientHeartColor provides Color.Red
    ) {
        AmbientHeartColor.current // I can access color here
    }
}
```

```
val AmbientHeartColor = ambientOf<Color>()

@Composable
fun HeartsScreen() {
    Providers(
        AmbientHeartColor provides Color.Red
    ) {
        AmbientHeartColor.current // I can access color here
    }
}
```

```
val AmbientHeartColor = ambientOf<Color>()

@Composable
fun HeartsScreen() {
    Providers(
        AmbientHeartColor provides Color.Red
    ) {
        AmbientHeartColor.current // I can access color here
    }
}
```

```
val AmbientHeartColor = ambientOf<Color>()

@Composable
fun HeartsScreen() {
    Providers(
        AmbientHeartColor provides Color.Red
    ) {
        AmbientHeartColor.current // I can access color here
    }
}
```

Key	Value
AmbientHeart	Color.Red
ContextAmbient	Context
LifecycleOwnerAmbient	LifecycleOwner

`ambientOf<Color>()`

VS

`staticAmbientOf<Color>()`

```
val AmbientTypography = staticAmbientOf<Typography>()

@Composable
val TextStyles: Typography
    get() = AmbientTypography.current

TextStyles.H1 // AmbientTypography.current.H1
```

```
val AmbientTypography = staticAmbientOf<Typography>()

@Composable
val TextStyles: Typography
    get() = AmbientTypography.current

TextStyles.H1 // AmbientTypography.current.H1
```

```
val AmbientTypography = staticAmbientOf<Typography>()

@Composable
val TextStyles: Typography
    get() = AmbientTypography.current

TextStyles.H1 // AmbientTypography.current.H1
```

```
val AmbientTypography = staticAmbientOf<Typography>()

@Composable
val TextStyles: Typography
    get() = AmbientTypography.current

TextStyles.H1 // AmbientTypography.current.H1
```

```
@Composable
fun Theme(
    typography: Typography,
    colors: Palette,
    dimensions: DimensionsTokens,
    content: @Composable () → Unit
) {
    Providers(
        AmbientTypography provides typography,
        AmbientColors provides colors,
        AmbientDimensions provides dimensions,
    ) {
        content()
    }
}
```

```
@Composable
fun Theme(
    typography: Typography,
    colors: Palette,
    dimensions: DimensionsTokens,
    content: @Composable () → Unit
) {
    Providers(
        AmbientTypography provides typography,
        AmbientColors provides colors,
        AmbientDimensions provides dimensions,
    ) {
        content()
    }
}
```

```
@Composable
fun Theme(
    typography: Typography,
    colors: Palette,
    dimensions: DimensionsTokens,
    content: @Composable () → Unit
) {
    Providers(
        AmbientTypography provides typography,
        AmbientColors provides colors,
        AmbientDimensions provides dimensions,
    ) {
        content()//TextStyles.H1
    }
}
```

Ambients is not a DI framework!

1. Реализация дизайн-системы
   - Токены
   - Компоненты
   - Паттерны
2. Интеграция с Android View
3. Тестирование
4. Внедрение

MatchPhotos

You have a new match!

Hugo matched with you while you were away. Now’s the perfect time to send them a message

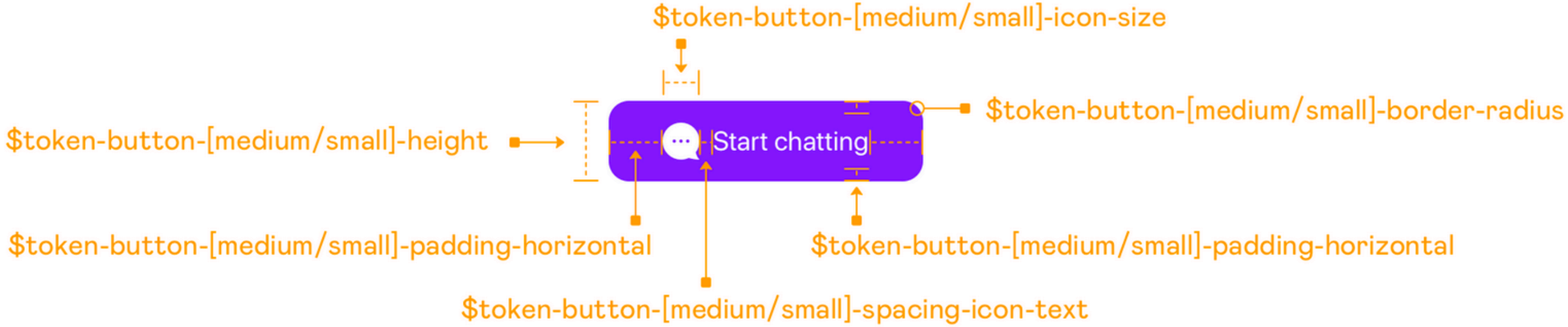
Button - Filled - White

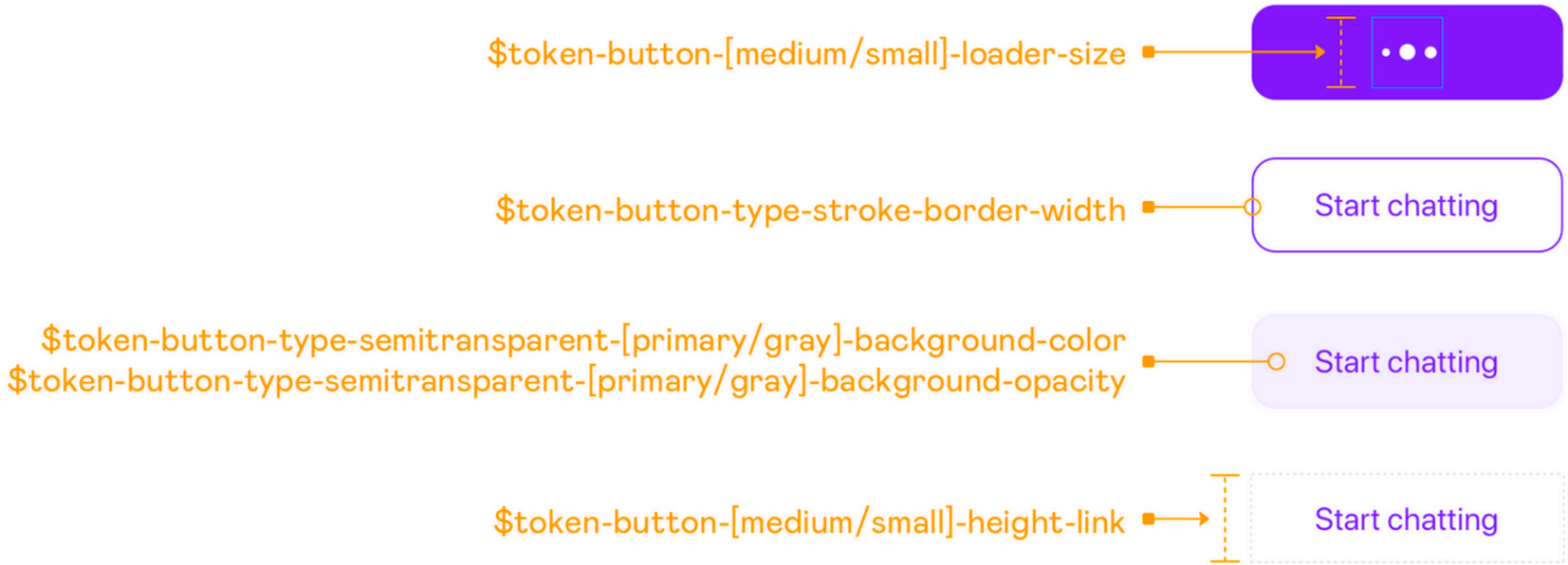
Send Message

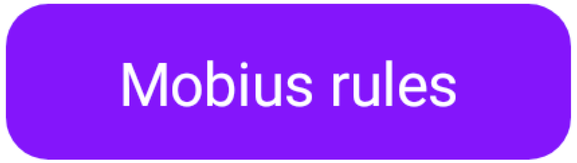
Button

```
@Composable
fun MaterialButton() {
    Button(
        onClick = {},
        content = {
            Text(text = "Hello Mobius”)
        }
    )
}
```

```
@Composable
fun SlotApi() {
    Button(
        onClick = {},
        backgroundColor = Color.Red
    ) {
        Button(
            onClick = {},
            backgroundColor = Color.Green
        ) {
            Button(
                onClick = {},
                backgroundColor = Color.Blue
            ) {
                Text(text = "We need to go deeper")
            }
        }
    }
}
```


$token-button-[medium/small]-icon-size
$token-button-[medium/small]-height
Start chatting
$token-button-[medium/small]-border-radius
$token-button-[medium/small]-padding-horizontal
$token-button-[medium/small]-padding-horizontal
$token-button-[medium/small]-spacing-icon-text


$token-button-[medium/small]-loader-size
$token-button-type-stroke-border-width
Start chatting
$token-button-type-semitransparent-[primary/gray]-background-color
$token-button-type-semitransparent-[primary/gray]-background-opacity
Start chatting
$token-button-[medium/small]-height-link
Start chatting

Filled	Stroke	Transparent
		
		

```
@Composable
fun Button(
    onClick: () → Unit,
    text: String,
    modifier: Modifier = Modifier,
    color: Color = Colors.primary,
    contentColor: Color = Color.White,
    icon: VectorAsset? = null,
    enabled: Boolean = true
)
```

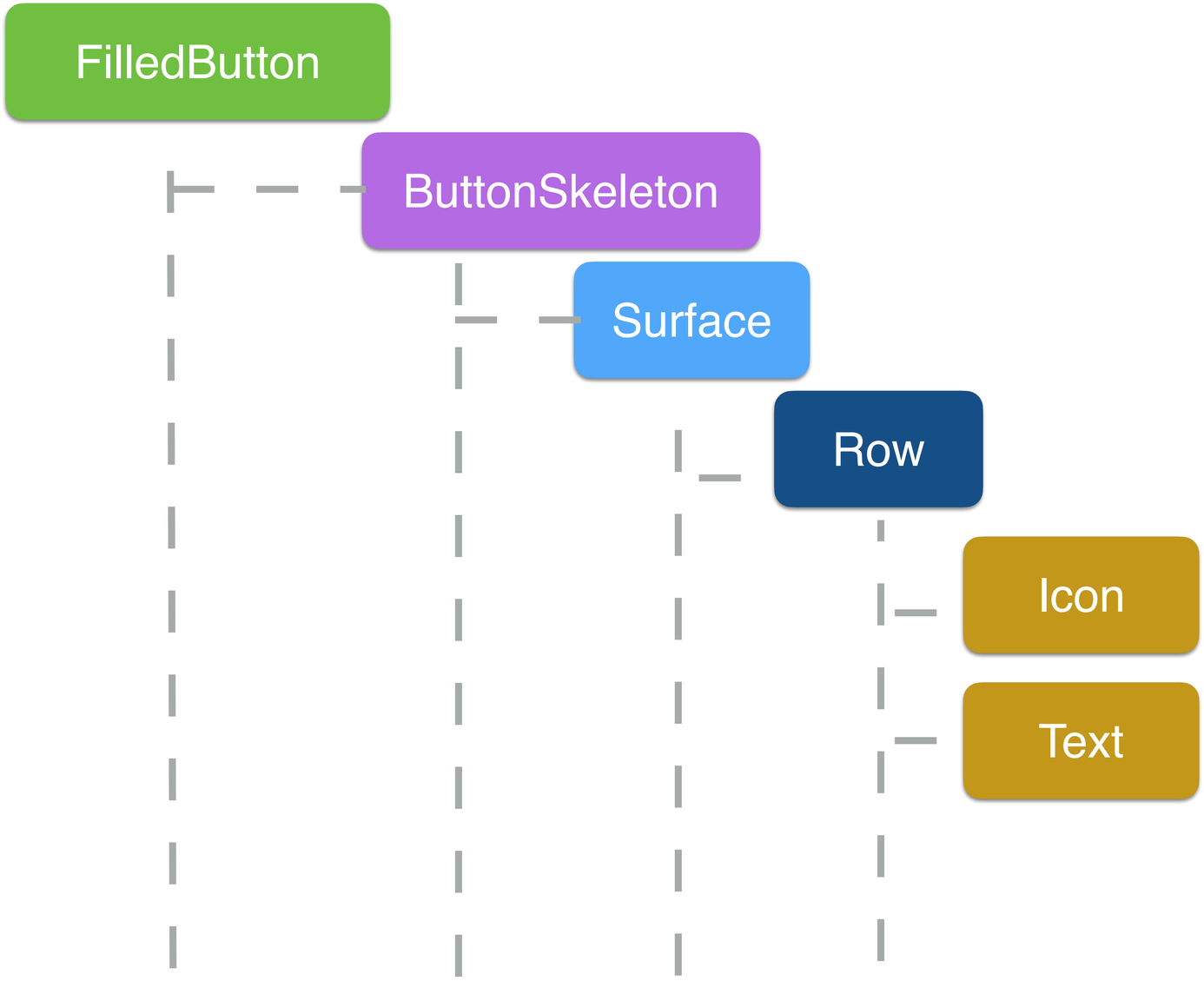
```
@Composable
fun Button(
    onClick: () → Unit,
    text: String,
    modifier: Modifier = Modifier,
    color: Color = Colors.primary,
    contentColor: Color = Color.White,
    icon: VectorAsset? = null,
    enabled: Boolean = true
)
```

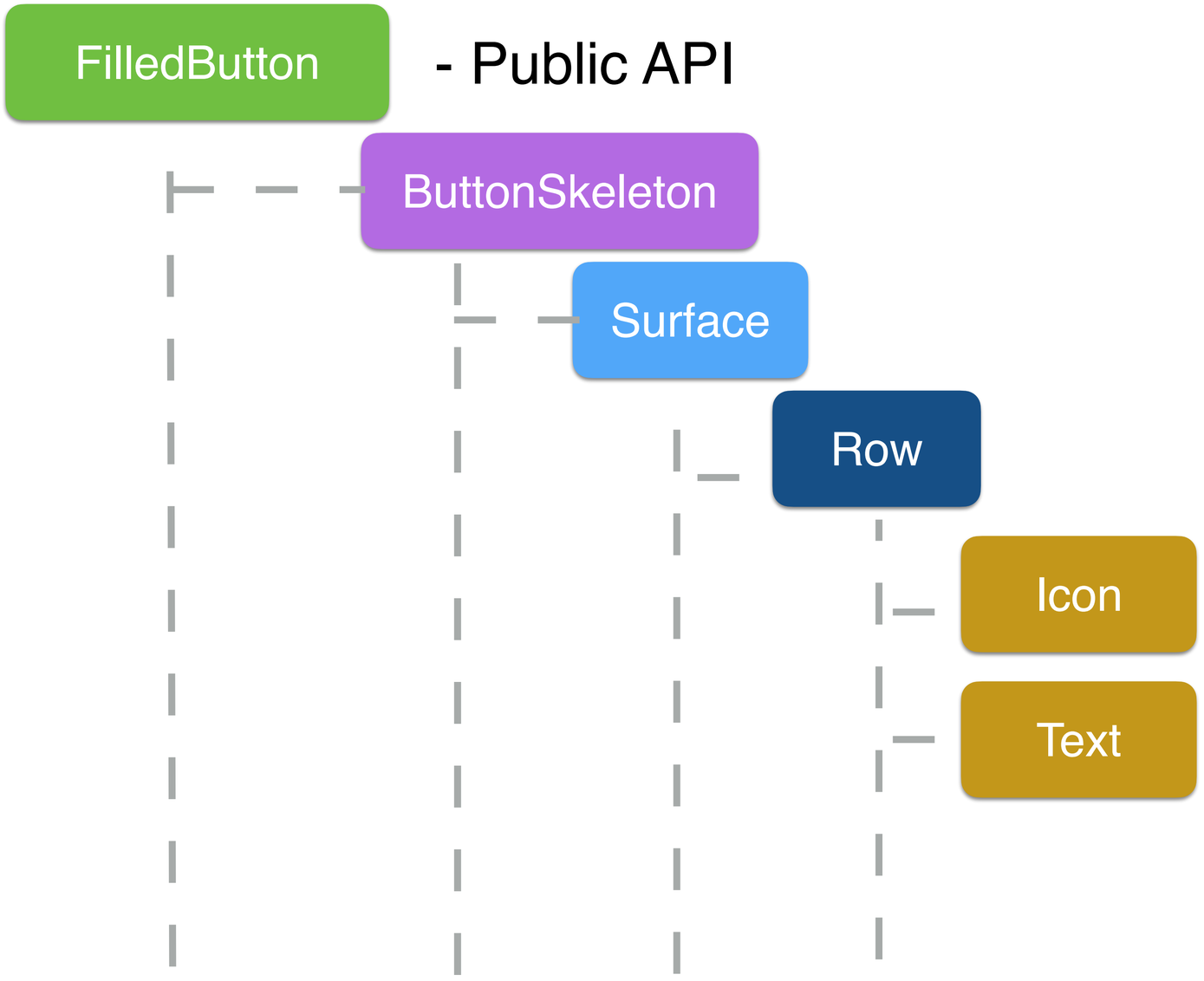
```
@Composable
fun Button(
    onClick: () → Unit,
    text: String,
    modifier: Modifier = Modifier,
    color: Color = Colors.primary,
    contentColor: Color = Color.White,
    icon: VectorAsset? = null,
    enabled: Boolean = true
)
```

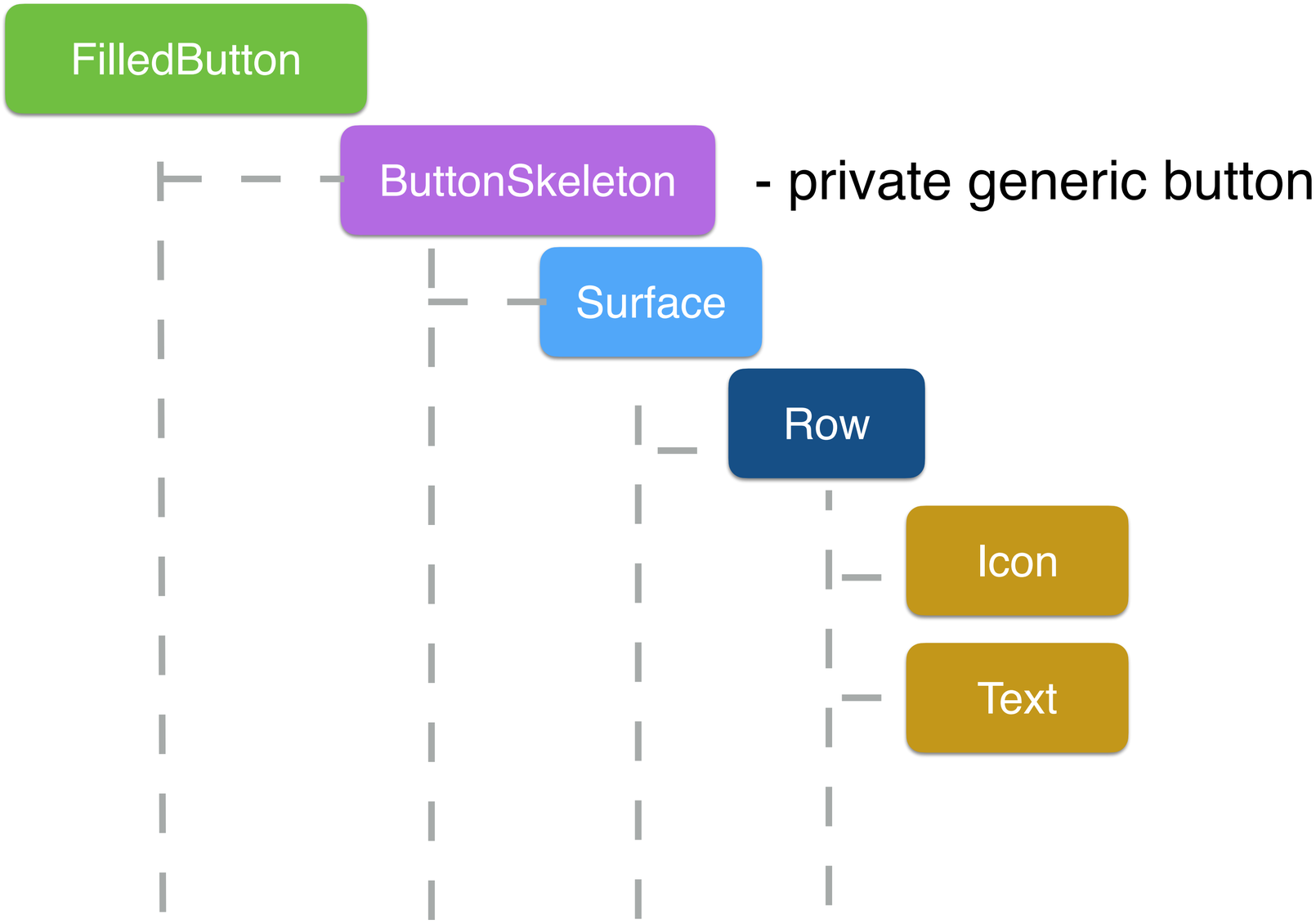
```
@Composable
fun Button(
    onClick: () → Unit,
    modifier: Modifier = Modifier,
    color: Color = Colors.primary,
    contentColor: Color = Color.White,
    icon: VectorAsset? = null,
    enabled: Boolean = true
)
```

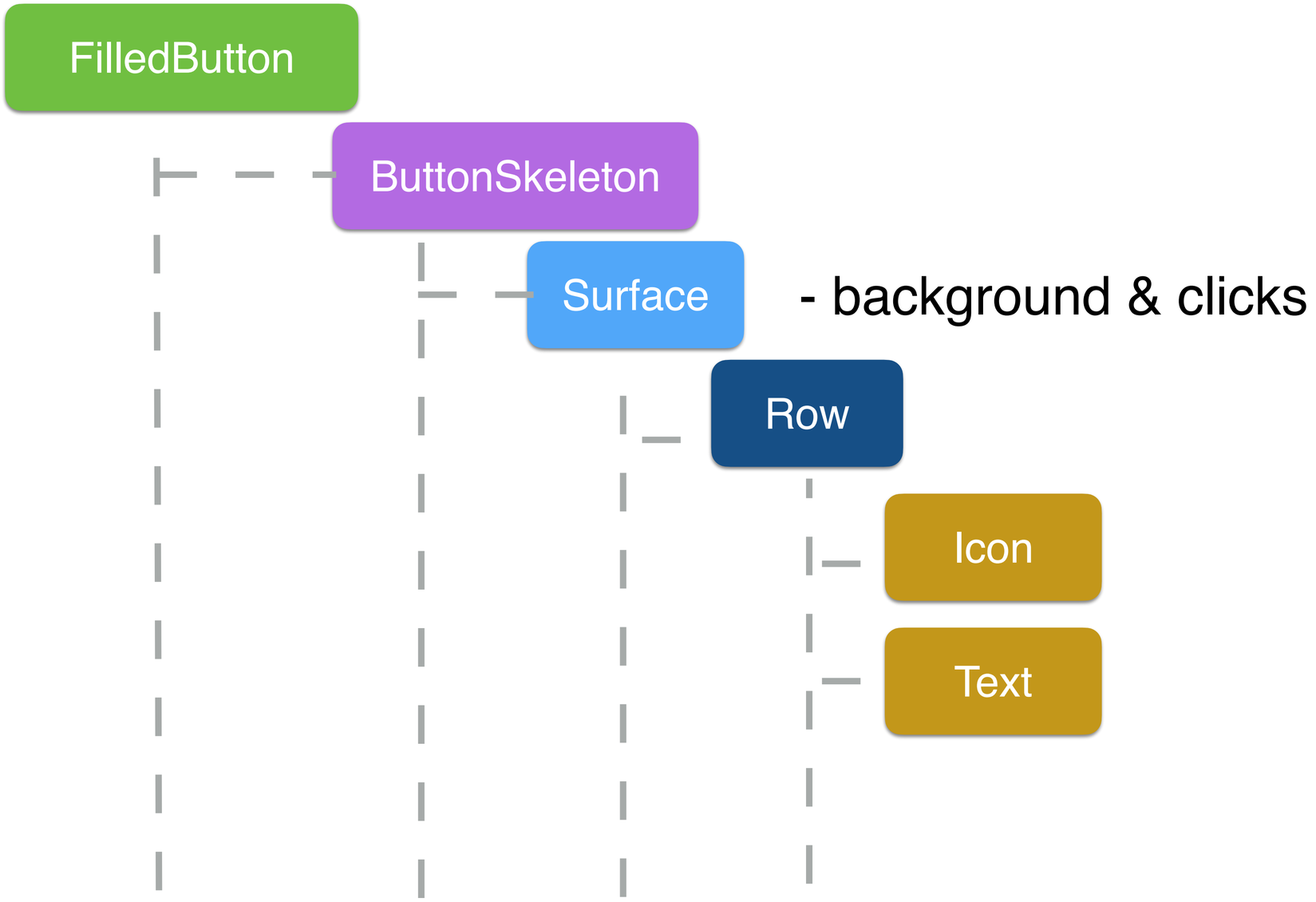
```
@Composable
private fun ButtonSkeleton(
    onClick: () → Unit,
    text: String,
    modifier: Modifier = Modifier,
    enabled: Boolean = true,
    shape: Shape = RoundedCornerShape(Dimensions.tokenButtonCornerRadius),
    border: BorderStroke? = null,
    backgroundColor: Color,
    contentColor: Color = Color.White,
    contentPadding: PaddingValues = PaddingValues(
        start = Dimensions.tokenButtonPaddingHorizontal,
        end = Dimensions.tokenButtonPaddingHorizontal,
        top = Dimensions.tokenButtonPaddingVertical,
        bottom = Dimensions.tokenButtonPaddingVertical
    ),
    height: Dp = Dimensions.tokenButtonHeight,
    icon: VectorAsset?,
)
```

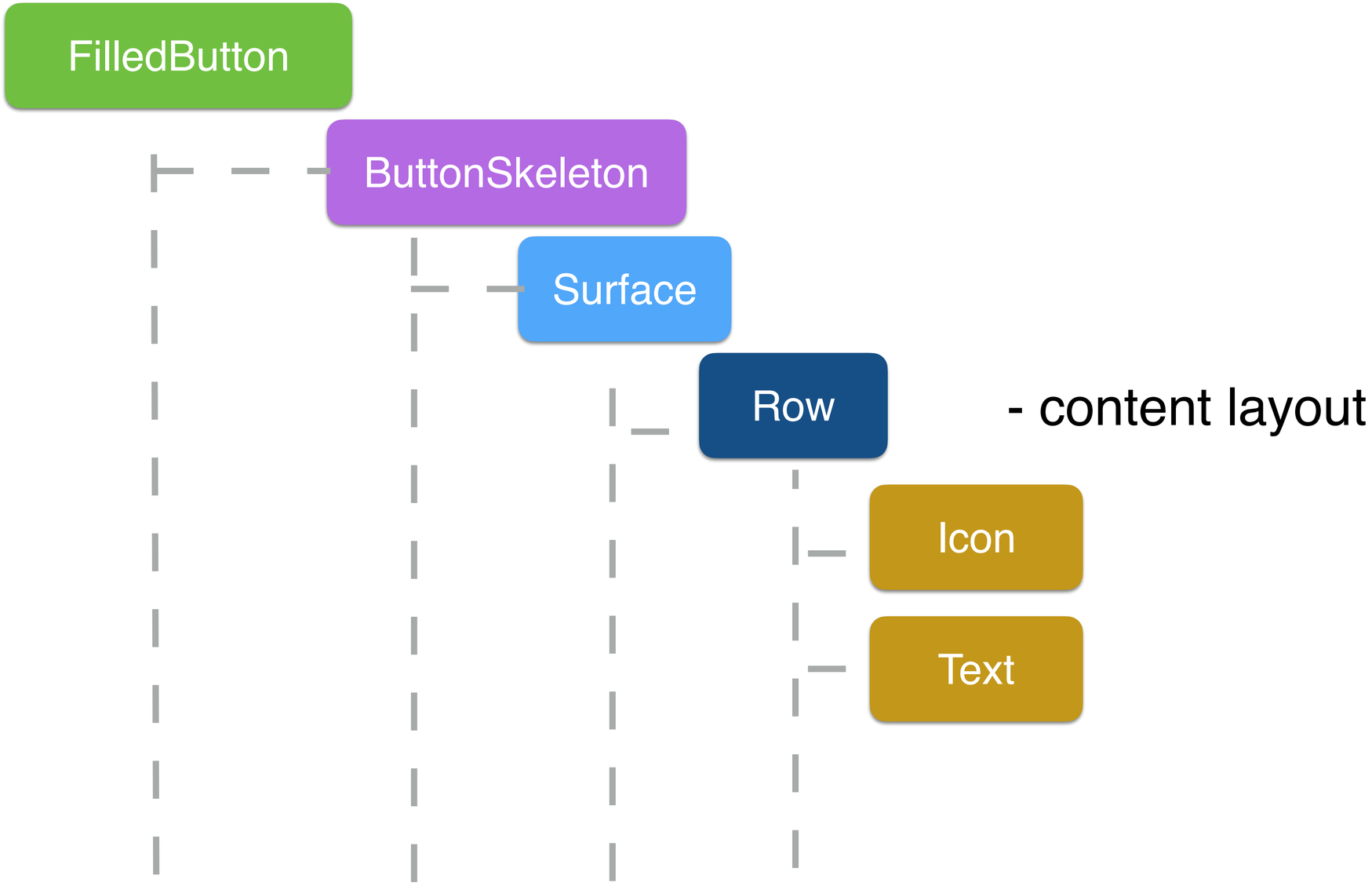
```
@Composable
private fun ButtonSkeleton(
    onClick: () → Unit,
    text: String,
    modifier: Modifier = Modifier,
    enabled: Boolean = true,
    shape: Shape = RoundedCornerShape(Dimensions.tokenButtonCornerRadius),
    border: BorderStroke? = null,
    backgroundColor: Color,
    contentColor: Color = Color.White,
    contentPadding: PaddingValues = PaddingValues(
        start = Dimensions.tokenButtonPaddingHorizontal,
        end = Dimensions.tokenButtonPaddingHorizontal,
        top = Dimensions.tokenButtonPaddingVertical,
        bottom = Dimensions.tokenButtonPaddingVertical
    ),
    height: Dp = Dimensions.tokenButtonHeight,
    icon: VectorAsset?
)
```

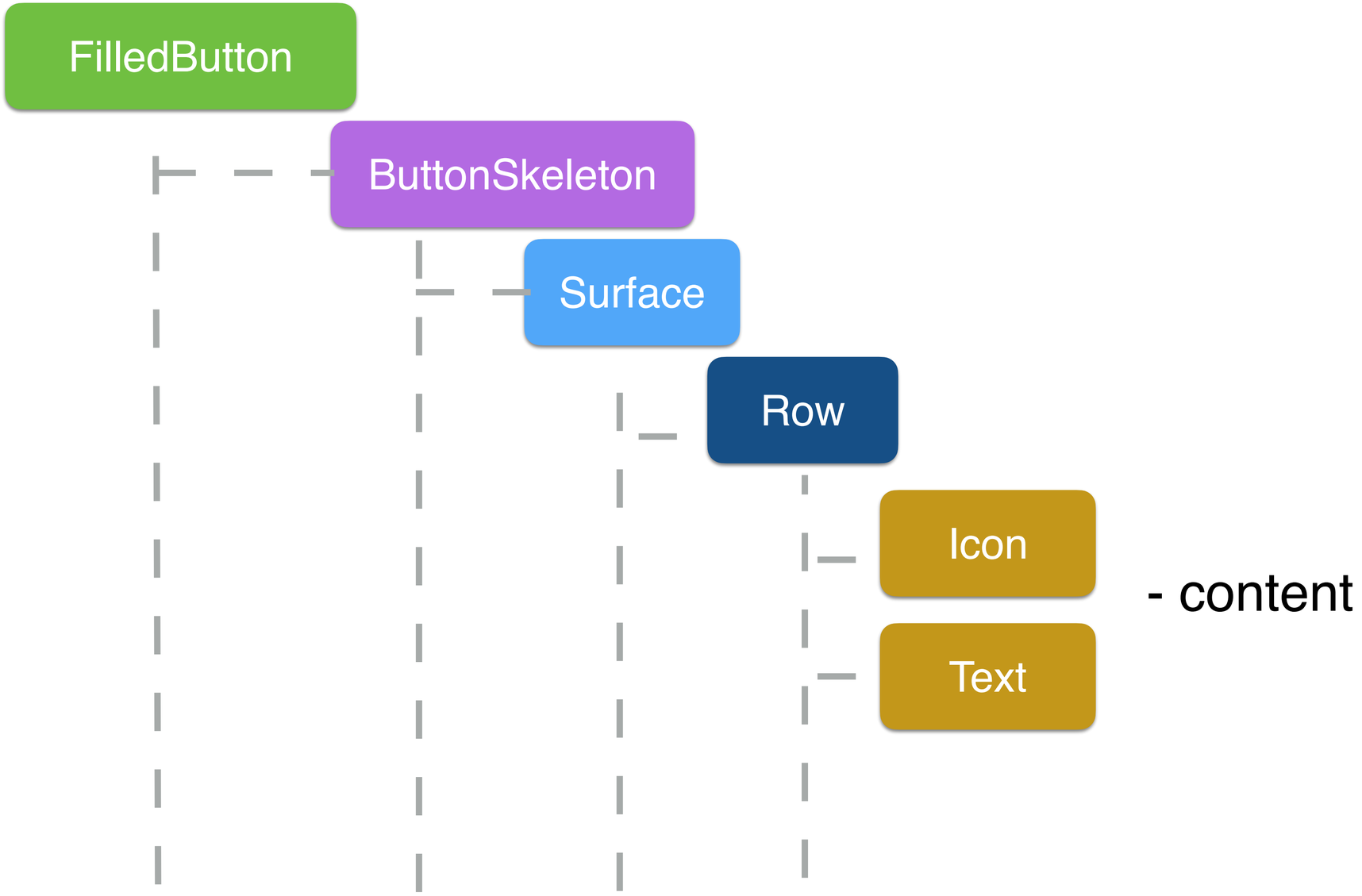

FilledButton
ButtonSkeleton
Surface
Row
Icon
Text


FilledButton
- Public API
ButtonSkeleton
Surface
Row
Icon
Text


FilledButton
ButtonSkeleton
- private generic button
Surface
Row
Icon
Text


FilledButton
ButtonSkeleton
Surface
- background & clicks
Row
Icon
Text


FilledButton
ButtonSkeleton
Surface
Row
- content layout
Icon
Text


FilledButton
ButtonSkeleton
Surface
Row
Icon
Text
- content

```
Surface(
    shape = shape,
    color = backgroundColor,
    contentColor = contentColor,
    border = border,
    modifier =
        modifier.clickable(
            onClick = onClick,
            enabled = enabled
        ).drawOpacity(
            if (enabled) 1f else Dimensions.tokenButtonDisabledOpacity
        )
)
```

```
Surface(
    shape = shape,
    color = backgroundColor,
    contentColor = contentColor,
    border = border,
    modifier =
        modifier.clickable(
            onClick = onClick,
            enabled = enabled
        ).drawOpacity(
            if (enabled) 1f else Dimensions.tokenButtonDisabledOpacity
        )
)
```

```
Surface(
    shape = shape,
    color = backgroundColor,
    contentColor = contentColor,
    border = border,
    modifier =
        modifier.clickable(
            onClick = onClick,
            enabled = enabled
        ).drawOpacity(
            if (enabled) 1f else Dimensions.tokenButtonDisabledOpacity
        )
)
```

```
Surface(
    shape = shape,
    color = backgroundColor,
    contentColor = contentColor,
    border = border,
    modifier =
        modifier.clickable(
            onClick = onClick,
            enabled = enabled
        ).drawOpacity(
            if (enabled) 1f else Dimensions.tokenButtonDisabledOpacity
        )
)
```

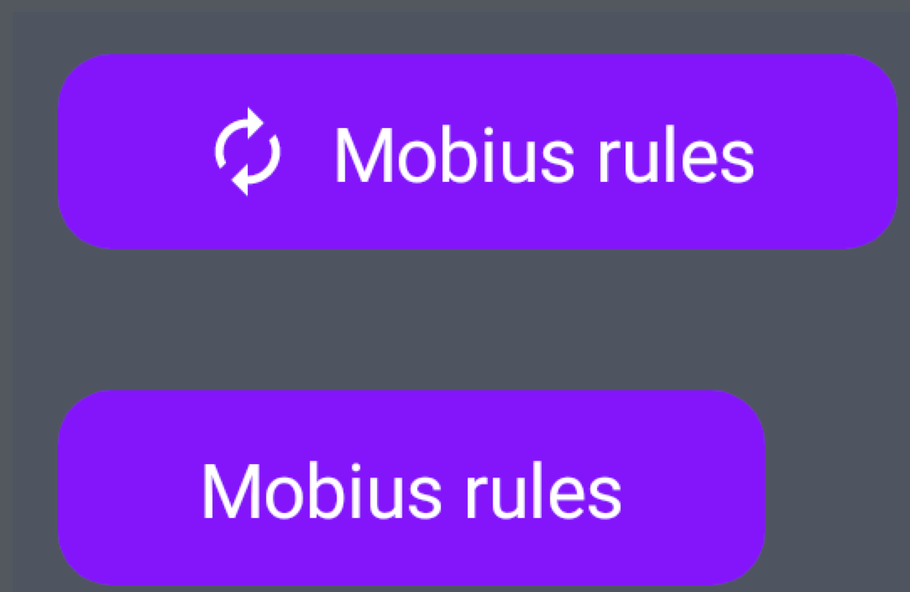
```
Row(
    Modifier
        .defaultMinSizeConstraints(minHeight = height)
        .padding(contentPadding),
    horizontalArrangement = Arrangement.Center,
    verticalAlignment = Alignment.CenterVertically,
)
```

```
if (icon ≠ null) {
    Icon(
        modifier = Modifier.size(Dimensions.tokenButtonIconSize),
        asset = icon,
        tint = contentColor
    )
    Spacer(modifier = Modifier.width(Dimensions.tokenButtonIconTextSpacing))
}
Text(
    text = text,
    style = TextStyles.Action,
    color = contentColor,
    maxLines = 1,
    overflow = TextOverflow.Ellipsis
)
```

```
if (icon ≠ null) {
    Icon(
        modifier = Modifier.size(Dimensions.tokenButtonIconSize),
        asset = icon,
        tint = contentColor
    )
    Spacer(modifier = Modifier.width(Dimensions.tokenButtonIconTextSpacing))
}
Text(
    text = text,
    style = TextStyles.Action,
    color = contentColor,
    maxLines = 1,
    overflow = TextOverflow.Ellipsis
)
```

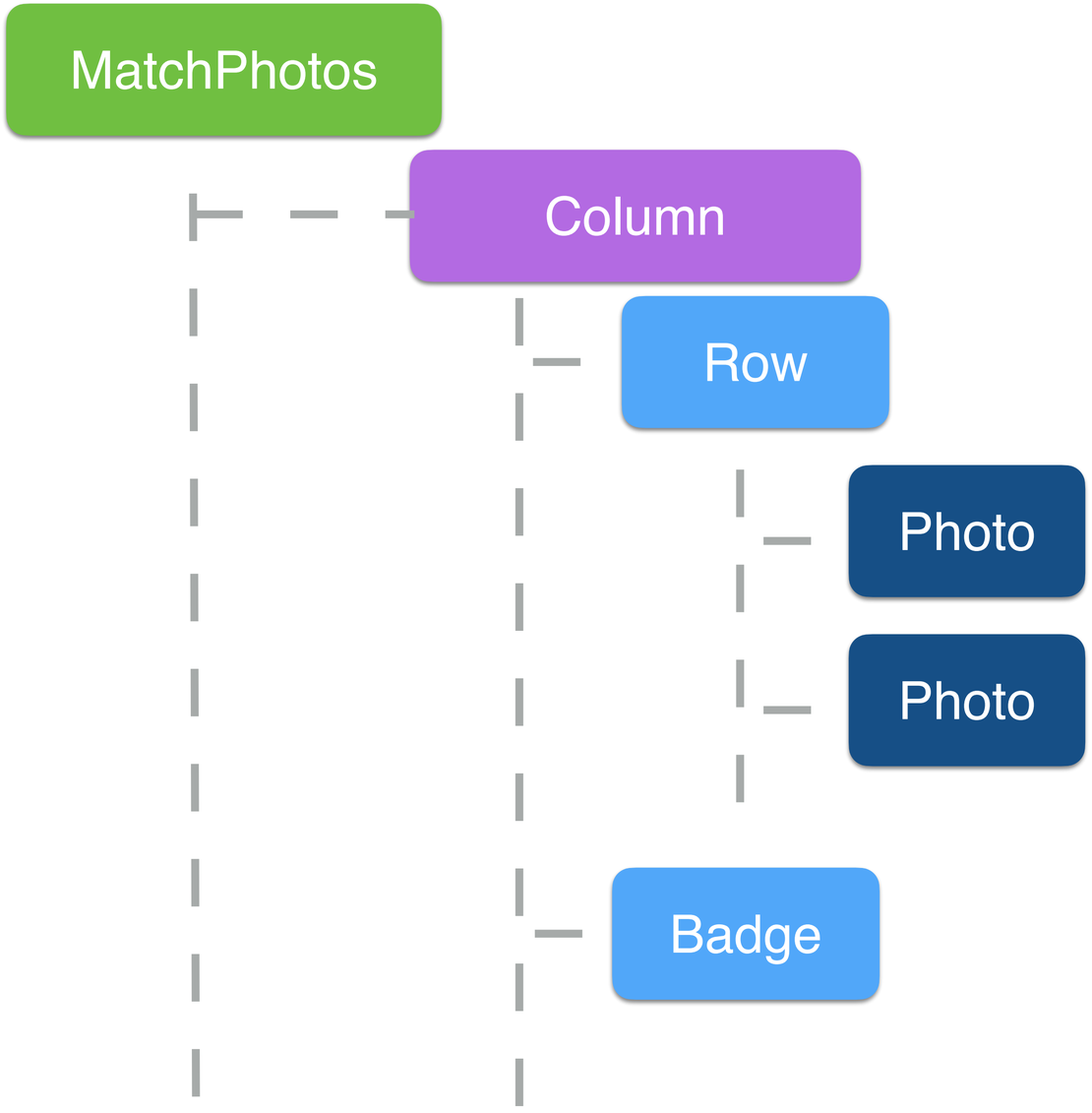
```
if (icon ≠ null) {
    Icon(
        modifier = Modifier.size(Dimensions.tokenButtonIconSize),
        asset = icon,
        tint = contentColor
    )
    Spacer(modifier = Modifier.width(Dimensions.tokenButtonIconTextSpacing))
}
Text(
    text = text,
    style = TextStyles.Action,
    color = contentColor,
    maxLines = 1,
    overflow = TextOverflow.Ellipsis
)
```

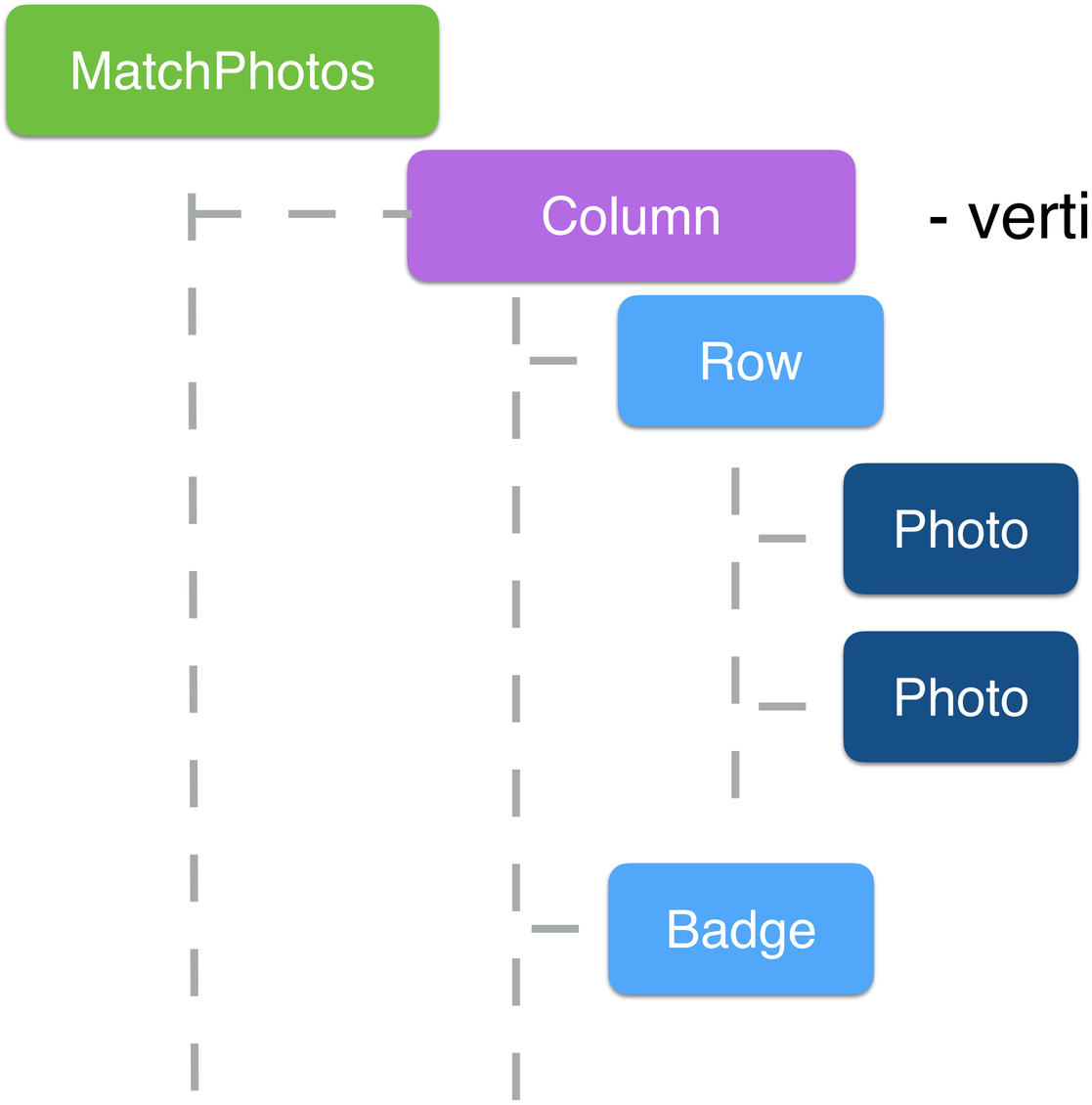
```kotlin
@Composable
fun Button(
    onClick: () -> Unit,
    text: String,
    modifier: Modifier = Modifier,
    color: Color = Colors.primary,
    contentColor: Color = Color.White,
    icon: VectorAsset? = null,
    enabled: Boolean = true
) {
    ButtonSkeleton(
        onClick = onClick,
        modifier = modifier,
        backgroundColor = color,
        contentColor = contentColor,
        enabled = enabled,
        icon = icon,
        text = text
    )
}
```



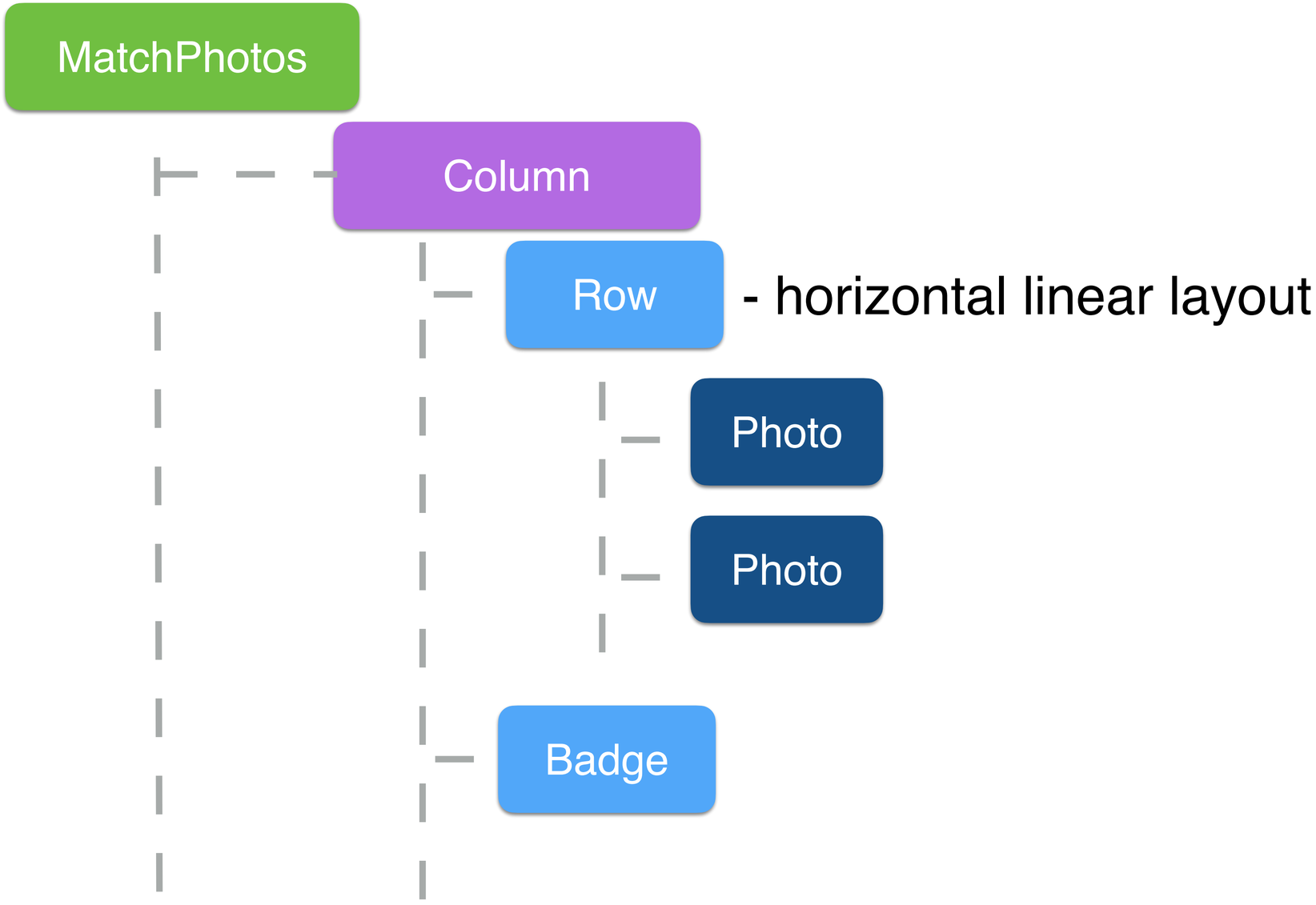
MatchPhotos

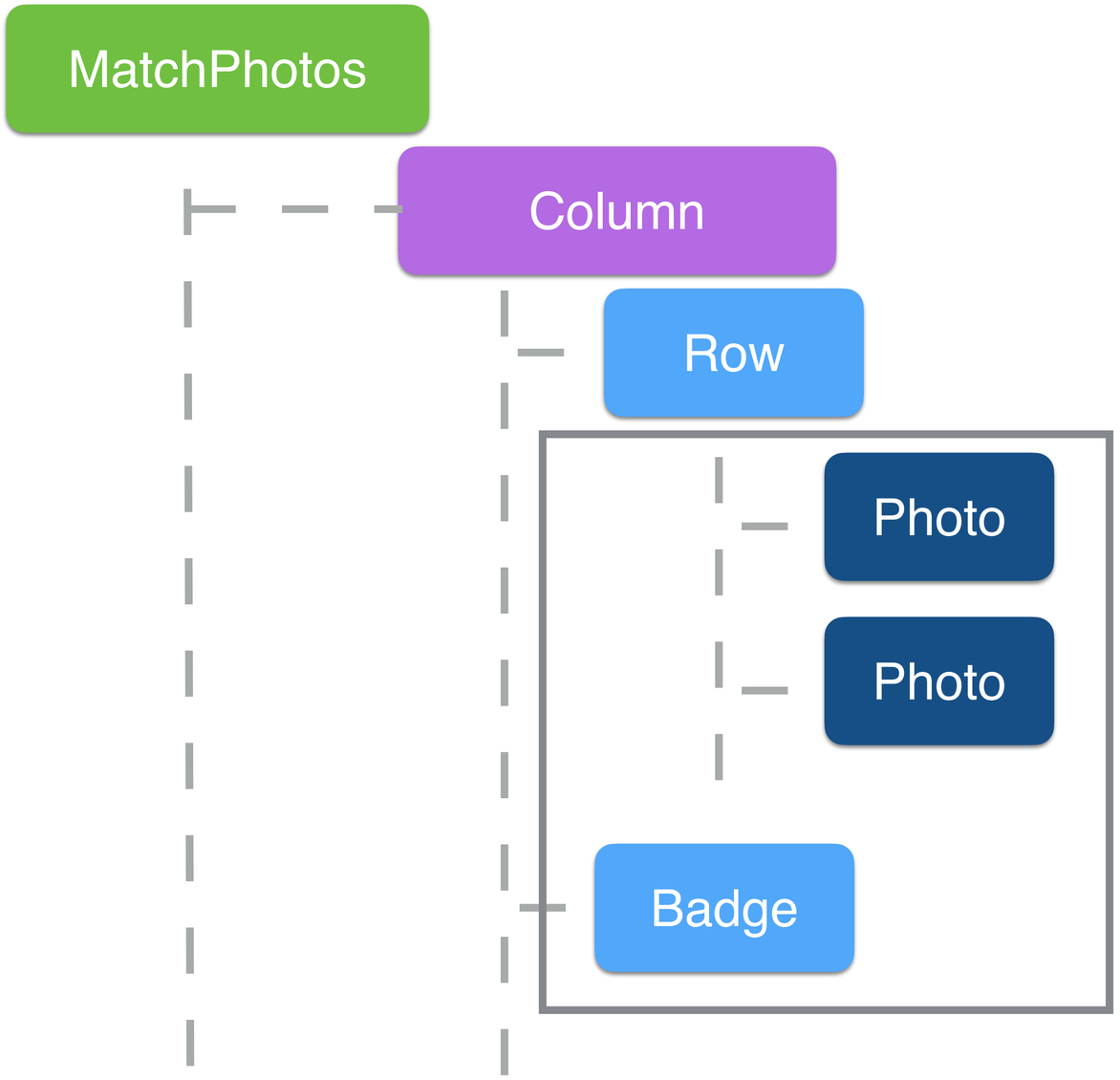
```
@Composable
fun MatchPhotos(
    leftPhoto: ImageAsset,
    rightPhoto: ImageAsset,
    badge: VectorAsset,
    modifier: Modifier = Modifier
)
```


MatchPhotos
Column
Row
Photo
Photo
Badge



- vertical linear layout


MatchPhotos
Column
Row
- horizontal linear layout
Photo
Photo
Badge



- content

```
Image(
    asset = rightPhoto,
    modifier = Modifier
        .drawLayer(
            rotationZ = rotationDegrees,
            translationX = rightTranslateX,
            translationY = rightTranslateY
        )
        .zIndex(rightZIndex)
        .size(photoSize)
        .clip(photoShape)
        .border(
            width = borderWidth,
            Colors.matchPhotosBorder,
            photoShape
        )
)
```

```
Image(
    asset = rightPhoto,
    modifier = Modifier
        .drawLayer(
            rotationZ = rotationDegrees,
            translationX = rightTranslateX,
            translationY = rightTranslateY
        )
        .zIndex(rightZIndex)
        .size(photoSize)
        .clip(photoShape)
        .border(
            width = borderWidth,
            Colors.matchPhotosBorder,
            photoShape
        )
)
```

```
Image(
    asset = rightPhoto,
    modifier = Modifier
        .drawLayer(
            rotationZ = rotationDegrees,
            translationX = rightTranslateX,
            translationY = rightTranslateY
        )
        .zIndex(rightZIndex)
        .size(photoSize)
        .clip(photoShape)
        .border(
            width = borderWidth,
            Colors.matchPhotosBorder,
            photoShape
        )
)
```

```
Image(
    asset = rightPhoto,
    modifier = Modifier
        .drawLayer(
            rotationZ = rotationDegrees,
            translationX = rightTranslateX,
            translationY = rightTranslateY
        )
        .zIndex(rightZIndex)
        .size(photoSize)
        .clip(photoShape)
        .border(
            width = borderWidth,
            Colors.matchPhotosBorder,
            photoShape
        )
)
```

```
Image(
    asset = rightPhoto,
    modifier = Modifier
        .drawLayer(
            rotationZ = rotationDegrees,
            translationX = rightTranslateX,
            translationY = rightTranslateY
        )
        .zIndex(rightZIndex)
        .size(photoSize)
        .clip(photoShape)
        .border(
            width = borderWidth,
            Colors.matchPhotosBorder,
            photoShape
        )
)
```

```
Row {
    Image(
        asset = leftPhoto,
        ...
    )
    Image(
        asset = rightPhoto,
        ...
    )
}
```

```
Column{
  Row {
      Image()
      Image()
  }
  Image(
      asset = badge,
      modifier = Modifier
          .size(iconSize)
          .align(Alignment.CenterHorizontally)
          .drawShadow(
              elevation = badgeElevation,
              shape = CircleShape
          )
  )
}
```

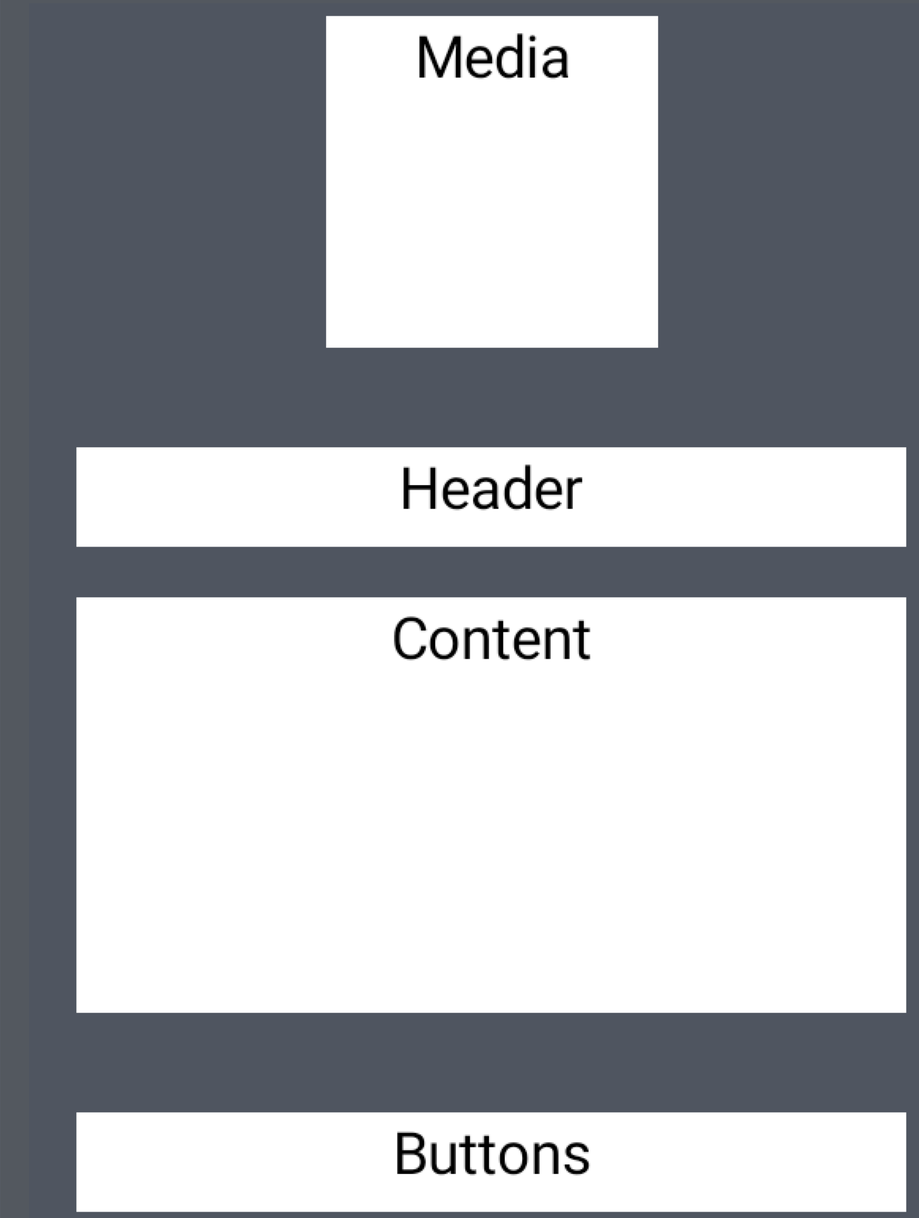
CtaBox

You have a new match!

Hugo matched with you while you were away. Now’s the perfect time to send them a message

Send Message

```
@Composable
fun CtaBox(
    media: @Composable () → Unit,
    header: @Composable () → Unit,
    content: @Composable () → Unit,
    buttons: @Composable() → Unit,
    modifier: Modifier = Modifier
)
```



```
Column(
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally,
        modifier = modifier.padding(Spacing.xlg)
    ) {
          media()
          Spacer(modifier = Modifier.size(Spacing.xlg))
          header()
          Spacer(modifier = Modifier.size(Spacing.md))
          content()
          Spacer(modifier = Modifier.size(Spacing.xlg))
          buttons()
    }
```

```
Column(
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally,
        modifier = modifier.padding(Spacing.xlg)
    ) {
          media()
          Spacer(modifier = Modifier.size(Spacing.xlg))
          header()
          Spacer(modifier = Modifier.size(Spacing.md))
          content()
          Spacer(modifier = Modifier.size(Spacing.xlg))
          buttons()
    }
```

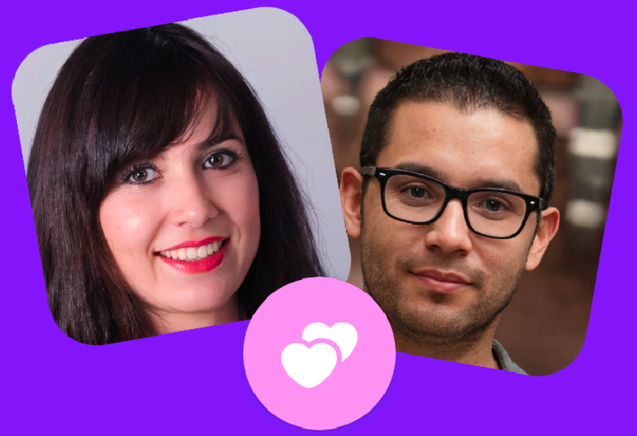
```
Column(
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally,
        modifier = modifier.padding(Spacing.xlg)
    ) {
        media()
        Spacer(modifier = Modifier.size(Spacing.xlg))
        header()
        Spacer(modifier = Modifier.size(Spacing.md))
        content()
        Spacer(modifier = Modifier.size(Spacing.xlg))
        buttons()
    }
```

```
Column(
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally,
        modifier = modifier.padding(Spacing.xlg)
    ) {
        media()
        Spacer(modifier = Modifier.size(Spacing.xlg))
        header()
        Spacer(modifier = Modifier.size(Spacing.md))
        content()
        Spacer(modifier = Modifier.size(Spacing.xlg))
        buttons()
    }
```

```
Column(
      verticalArrangement = Arrangement.Center,
      horizontalAlignment = Alignment.CenterHorizontally,
      modifier = modifier.padding(Spacing.xlg)
    ) {
        media()
        Spacer(modifier = Modifier.size(Spacing.xlg))
        header()
        Spacer(modifier = Modifier.size(Spacing.md))
        content()
        Spacer(modifier = Modifier.size(Spacing.xlg))
        buttons()
   }
```

```
Column(
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally,
        modifier = modifier.padding(Spacing.xlg)
    ) {
          media()
          Spacer(modifier = Modifier.size(Spacing.xlg))
          header()
          Spacer(modifier = Modifier.size(Spacing.md))
          content()
          Spacer(modifier = Modifier.size(Spacing.xlg))
          buttons()
    }
```

```
@Composable
fun MatchScreen(config: MatchConfig) {
      CtaBox(
          media = { MatchPhotos(..) },
          header = {
            Text(text="",style=TextStyles.H1)
          },
          content = {
            Text(text="",style=TextStyles.P1)
          },
          buttons = {
              Button(
                  text = "...",
                  color = Color.White,
                  contentColor = Color.Black
              )
          }
      )
}
```



You have a new match!

Hugo matched with you while you were away. Now’s the perfect time to send them a message

Send Message

You have a new match!

Hugo matched with you while you were away. Now’s the perfect time to send them a message

Send Message

You have a new match!

Hugo matched with you while you were away. Now’s the perfect time to send them a message

Send Message

1. Реализация дизайн-системы
   - Токены
   - Компоненты
   - Паттерны
2. Интеграция с Android View
3. Тестирование
4. Внедрение

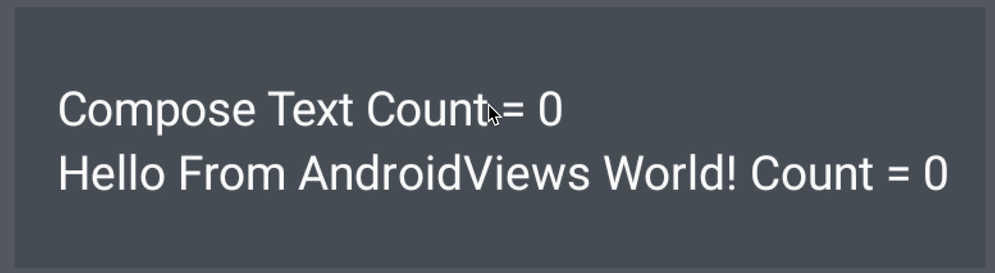
Android Views + Compose

```
val count = remember { mutableStateOf(0) }

Column {
    Text(text = "Compose Text Count = ${count.value}")
    AndroidView(
        viewBlock = { context →
            TextView(context).apply {
                setOnClickListener {
                    count.value = count.value + 1
                }
            }
        },
        update = { view →
            view.text = "AndroidViews World! Count = ${count.value}"
        }
    )
}
```

```
val count = remember { mutableStateOf(0) }

Column {
    Text(text = "Compose Text Count = ${count.value}")
    AndroidView(
        viewBlock = { context →
            TextView(context).apply {
                setOnClickListener {
                    count.value = count.value + 1
                }
            }
        },
        update = { view →
            view.text = "AndroidViews World! Count = ${count.value}"
        }
    )
}
```

```
val count = remember { mutableStateOf(0) }

Column {
    Text(text = "Compose Text Count = ${count.value}")
    AndroidView(
        viewBlock = { context →
            TextView(context).apply {
                setOnClickListener {
                    count.value = count.value + 1
                }
            }
        },
        update = { view →
            view.text = "AndroidViews World! Count = ${count.value}"
        }
    )
}
```

```
val count = remember { mutableStateOf(0) }

Column {
    Text(text = “Compose Count = ${count.value}")
    AndroidView(
        viewBlock = { context →
            TextView(context).apply {
                setOnClickListener {
                    count.value = count.value + 1
                }
            }
        },
        update = { view →
            view.text = "...Count = ${count.value}"
        }
    )
}
```



Compose + Android Views

```
<LinearLayout>

    <androidx.compose.ui.platform.ComposeView
        android:id="@+id/composeView"
        .../>

    <TextView
        android:id="@+id/androidView"
        .../>

</LinearLayout>
```

```
override fun onCreate() {

    val composeView = findViewById<ComposeView>(..)
    composeView.setContent {
        Text(text = "ComposeText")
    }

}
```

```
override fun onCreate() {

    val composeView = findViewById<ComposeView>(..)
    composeView.setContent {
        Text(text = "ComposeText")
    }

}
```

Тестирование

```
object TestTagMatchScreen {
    const val button = "MatchScreenButton"
    ...
}

Button(
    ...
    modifier = Modifier.testTag(TestTagMatchScreen.button)
)
```

```
@get:Rule
val composeTestRule = createComposeRule()

fun launchScreen() {
    composeTestRule.setContent {
        BadooTheme {
            MatchScreen()
        }
    }
}
```

```
@Test
fun matchScreenTest() {
    launchScreen()

    composeTestRule
            .onNodeWithTag(TestTagMatchScreen.button)
            .assertIsDisplayed()
            .assertTextEquals("Send Message")
            .performClick()
    }
}
```

```
@Test
fun interopTest() {
    launchScreen()

    onView(ViewMatchers.withContentDescription("CounterAndroidView"))
        .perform(click())
        .check(matches(withText("Hello From AndroidViews World! Count = 1")))

    composeTestRule
        .onNodeWithText(“Compose Text Count = 1").assertExists()
}
```

```
@Test
fun interopTest() {
    launchScreen()

    onView(ViewMatchers.withContentDescription("CounterAndroidView"))
        .perform(click())
        .check(matches(withText("Hello From AndroidViews World! Count = 1")))

    composeTestRule
        .onNodeWithText(“Compose Text Count = 1").assertExists()
}
```

```
@Test
fun interopTest() {
    launchScreen()

    onView(ViewMatchers.withContentDescription("CounterAndroidView"))
        .perform(click())
        .check(matches(withText("Hello From AndroidViews World! Count = 1")))

    composeTestRule
        .onNodeWithText(“Compose Text Count = 1").assertExists()
}
```

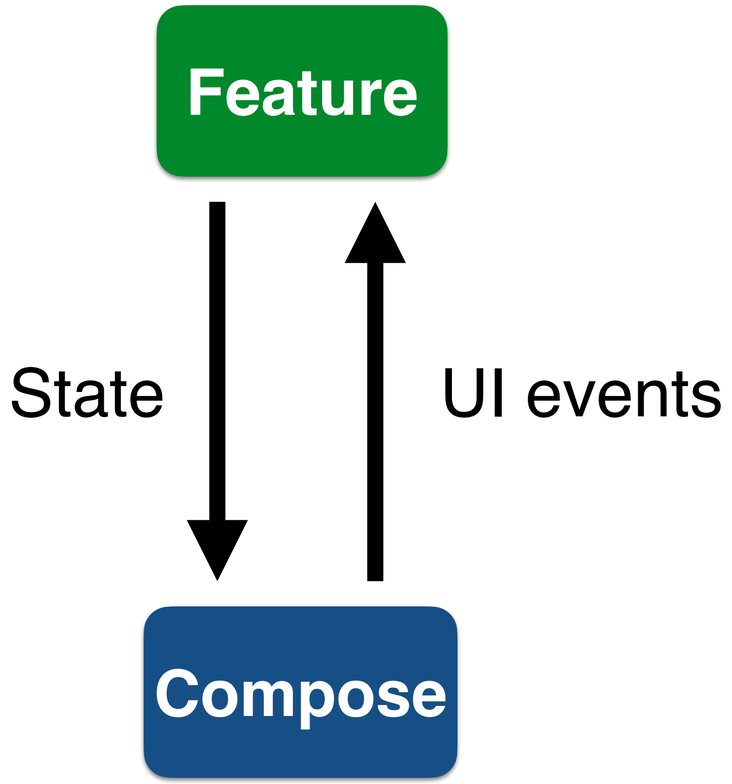
Скриншот тесты

Karumi — Shot

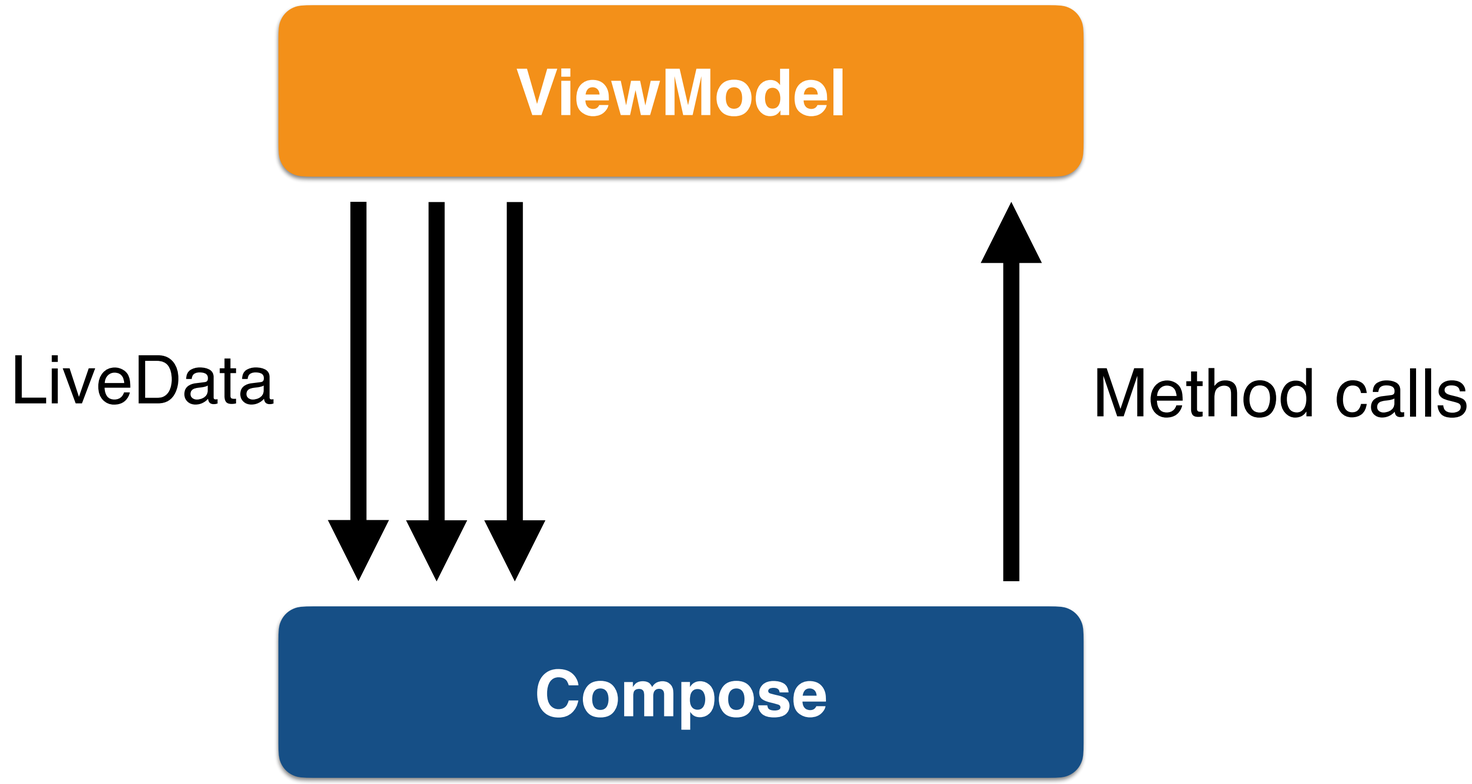
1. Реализация дизайн-системы
   - Токены
   - Компоненты
   - Паттерны
2. Интеграция с Android View
3. Тестирование
4. Внедрение

Внедрение в приложение

Интеграция с логикой — MVI



Интеграция с логикой — MVVM



```
val value: String by observable.subscribeAsState("initial")
val value: String by liveData.observeAsState(“initial")

val value: String? by observable.subscribeAsState(null)
val value: String? by liveData.observeAsState(null)


Text("Value is $value")
```

```
val value: String by observable.subscribeAsState("initial")
val value: String by liveData.observeAsState(“initial")

val value: String? by observable.subscribeAsState(null)
val value: String? by liveData.observeAsState(null)


Text("Value is $value")
```

```
val value: String by observable.subscribeAsState("initial")
val value: String by liveData.observeAsState("initial")

val value: String? by observable.subscribeAsState(null)
val value: String? by liveData.observeAsState(null)


Text("Value is $value")
```

```
val value: String by observable.subscribeAsState("initial")
val value: String by liveData.observeAsState("initial")

val value: String? by observable.subscribeAsState(null)
val value: String? by liveData.observeAsState(null)


Text("Value is $value")
```

Интеграция с логикой — MVP

Итог

✅ Как реализовать дизайн-систему?

✅ Как интегрировать с текущими экранами?

✅ Как тестировать?

✅ Как интегрировать с логикой?

Внедряем по шагам

- Набор базовых UI-компонентов
- Эксперименты
- Начинать с не критичных экранов
- Новые экраны
- Переписывать старое

Что дальше?

- Badoo + Compose
- Новые компоненты
- Ждем стабильную версию
- Отправляем в прод 🚀🚀🚀

Resources

- Design system talk
- Compose pathway
- Compse samples
- Repo for the talk

CHEERS!

antonshilov_

bumble badoo