Automating UI Infrastructure in Jetpack Compose

Vinay Gaba

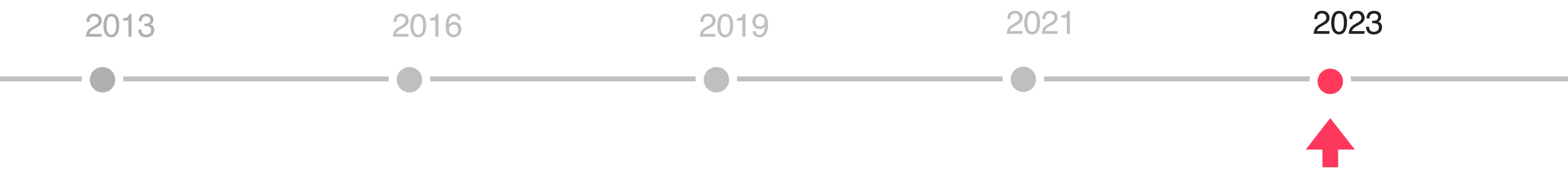
@VinayGaba

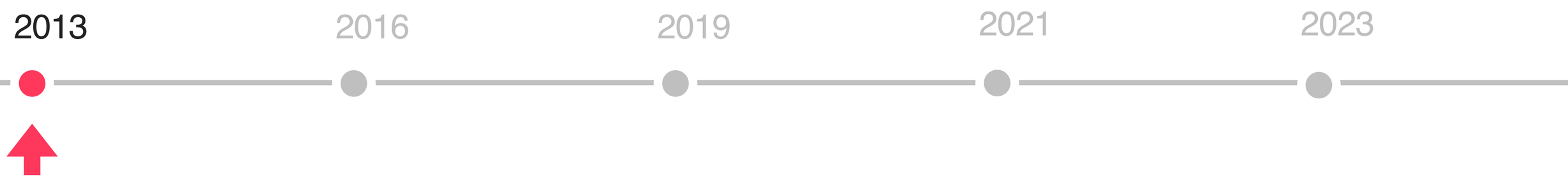
KotlinConf'23

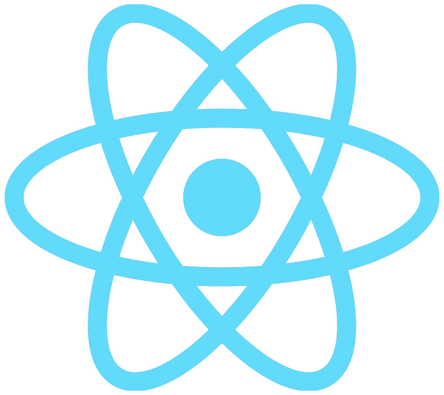
Amsterdam


airbnb

🚀 Automating UI Infrastructure 🚀


2013
2016
2019
2021
2023


2013
2016
2019
2021
2023



I don't know how to react to this #jsconf

2:21 PM · May 29, 2013

@facebook is announcing a new UI library called React at #jsconf. Some strange mix of XHTML and JS. Trying to wrap my head around it...

2:19 PM · May 29, 2013

Facebook React is not getting much love at #jsconf. Mixing HTML in JS seems like a bad idea

2:28 PM · May 29, 2013 from Jacksonville

Separation of concerns is a pretty core tenant of CS. React just seems wrongity wrong wrong wrong. #jsconf

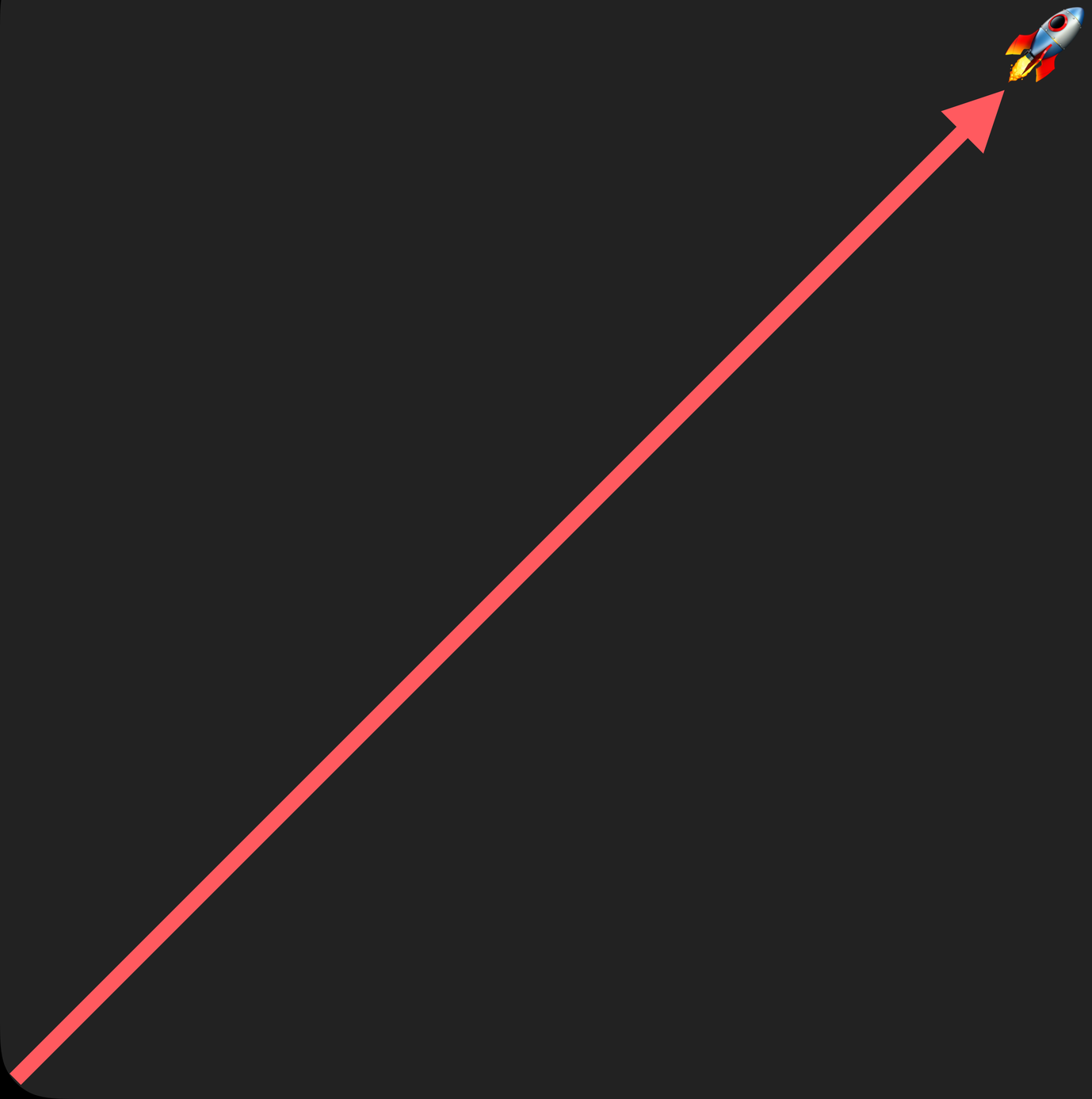
2:26 PM · May 29, 2013

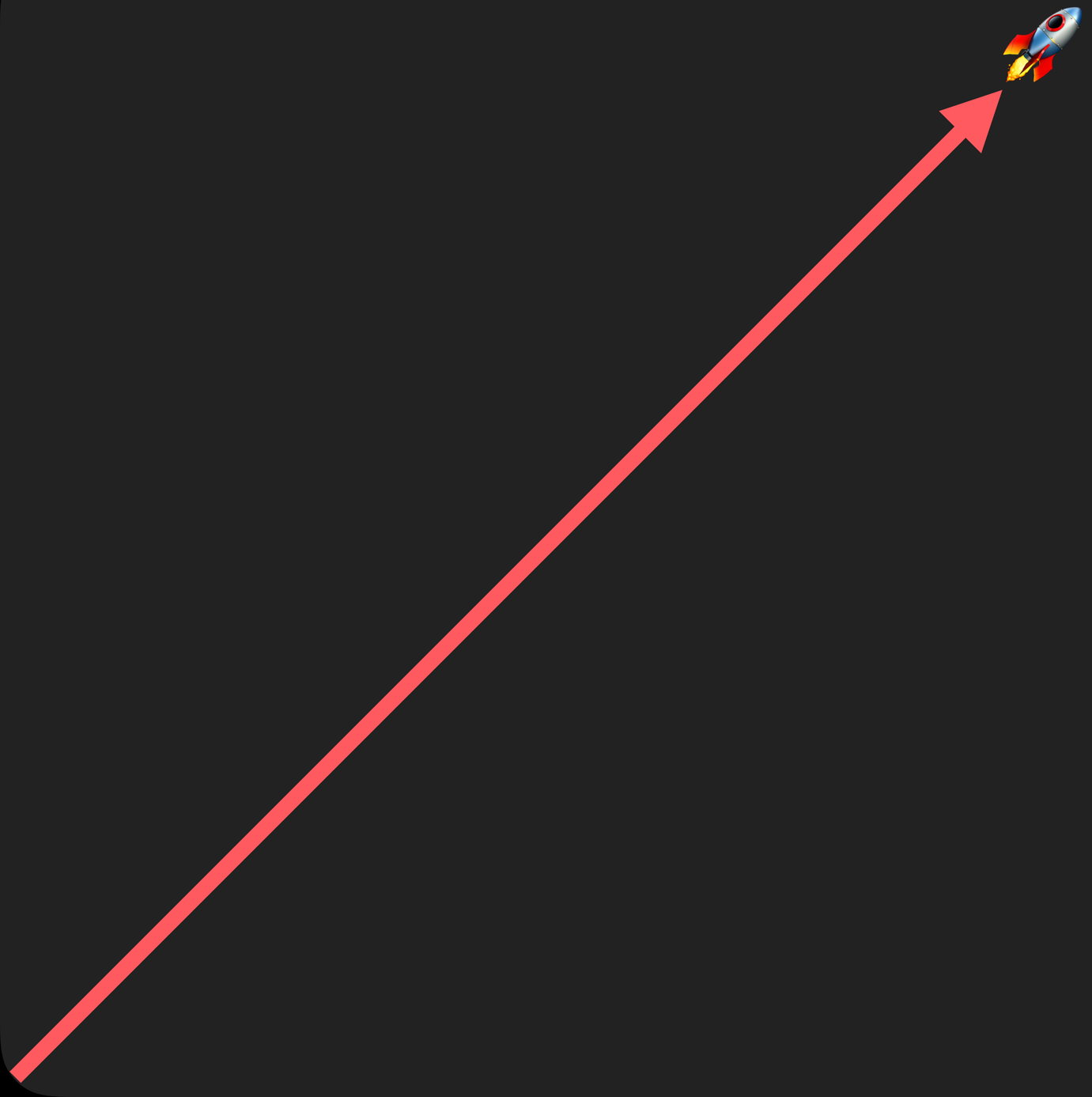
Wow, not good initial reactions to React.js from Facebook.... #jsconf

2:31 PM · May 29, 2013

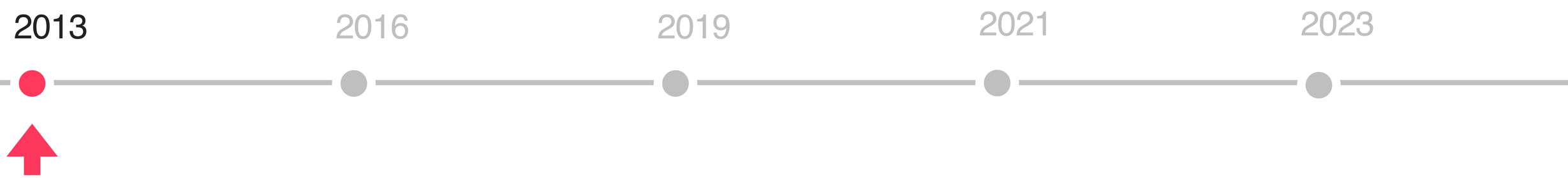
What!? #react!? XML markup in JS. Unsanity! It's *soo* new and innovative it's definitely wrong and bad! Very bad! lol@#jsconf.

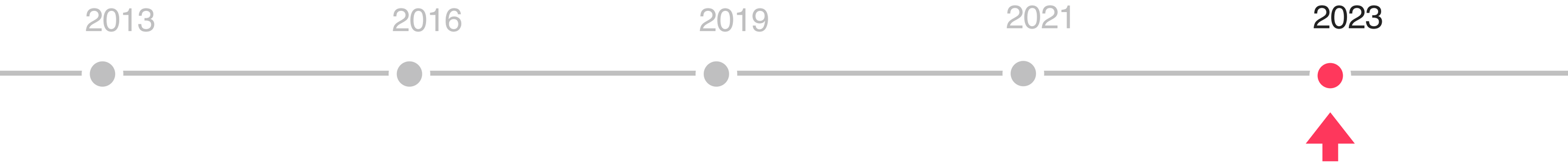
3:54 AM · May 30, 2013

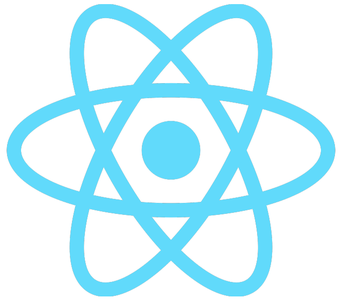


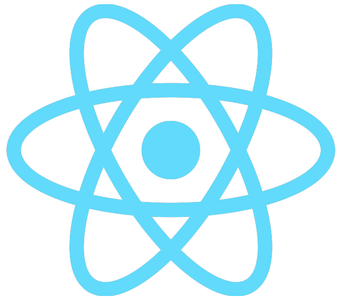


200k+
Github
Stars ⭐


2013
2016
2019
2021
2023


2013
2016
2019
2021
2023






android


iOS

Button

Optional title

Primary

Title that wraps >

Subtitle · Subtitle

This is a subheader

This is a description of the subheader that wraps to two or at the very most, three lines in this card.

Button

Button

Title that wraps >

Subtitle · Subtitle

Button

Button

Primary

Long Tertiary action

< Tertiary

5/6

Button >

```
fun MyCustomTextComponent(displayString: String) {

}
```

```
fun MyCustomTextComponent(displayString: String) {
    Text(
        text = displayString,
        style = TextStyle(
            fontWeight = FontWeight.W900,
            fontFamily = FontFamily.Monospace,
            fontSize = 32.sp,
            fontStyle = FontStyle.Italic,
            color = Color.Magenta,
            background = Color.Black
        )
    )
}
```

```
@Composable
fun MyCustomTextComponent(displayString: String) {
    Text(
        text = displayString,
        style = TextStyle(
            fontWeight = FontWeight.W900,
            fontFamily = FontFamily.Monospace,
            fontSize = 32.sp,
            fontStyle = FontStyle.Italic,
            color = Color.Magenta,
            background = Color.Black
        )
    )
}
```

```
@Composable
fun MyCustomTextComponent(displayString: String) {
    Text(
        text = displayString,
        style = TextStyle(
            fontWeight = FontWeight.W900,
            fontFamily = FontFamily.Monospace,
            fontSize = 32.sp,
            fontStyle = FontStyle.Italic,
            color = Color.Magenta,
            background = Color.Black
        )
    )
}
```

Primary wrapping text
Secondary wrapping text
Error text

Tertiary 5/6 Primary

Primary wrapping text
Secondary wrapping text

Wrapping Label Value

Button

Title text
Secondary Text

Address

Street address
City
State
Postal code

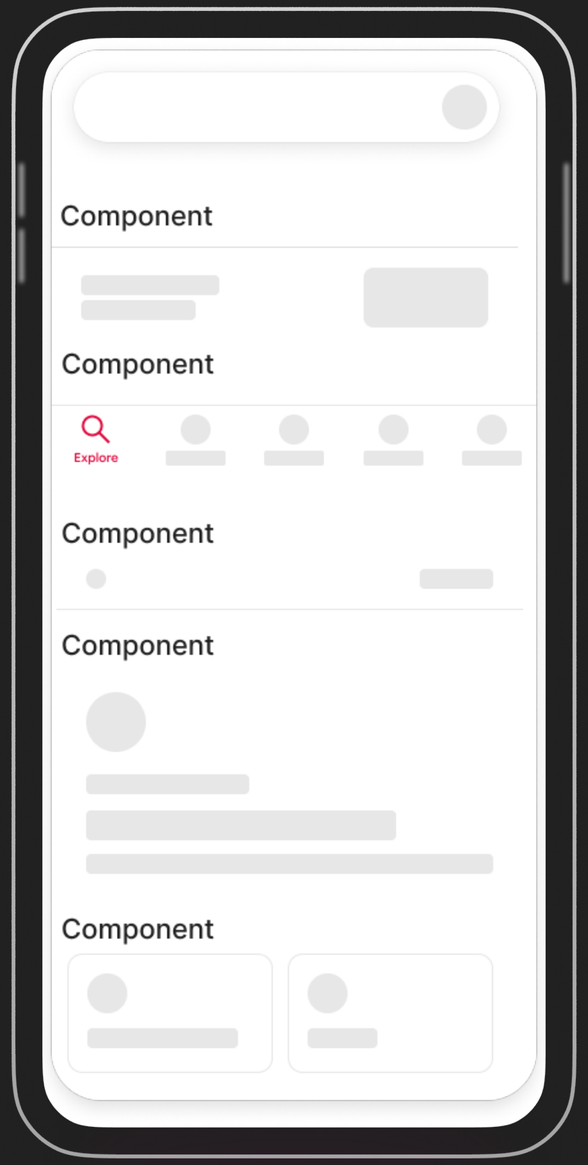
Optional Helper Text

Label
Input
Error text

This is an optional toast title with a max of two lines of copy
Body copy is mandatory with a limit of three lines. Consider a dialog for anything longer.

Action

Discoverability


Component
Component
Explore
Component
Component
Component

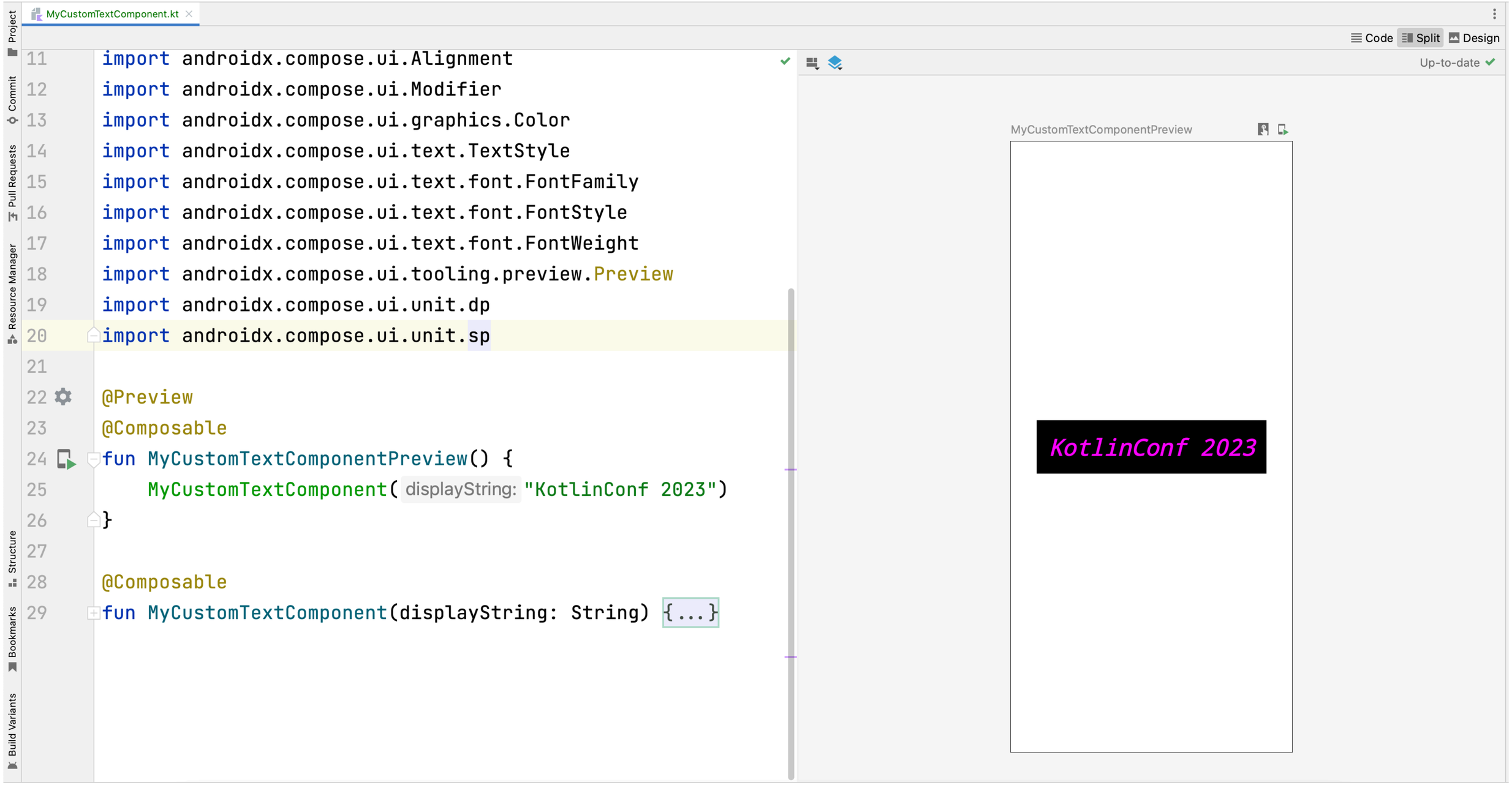
Project Codename: Showkase

Requirements

- 🌎 All components from the codebase
- 🗂️ Components are neatly organized
- 🙅 No manual work

```
@Composable
fun MyCustomTextComponent(displayString: String) {
    ...
}
```

```
@Composable
fun MyCustomTextComponent(displayString: String) {
    ...
}

@Preview(name = "MyCustomTextComponent", group = "SectionHeader")
@Composable
fun MyCustomTextComponentPreview() {
    MyCustomTextComponent("KotlinConf 2023")
}
```




```
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.text.font.FontStyle
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

@Preview
@Composable
fun MyCustomTextComponentPreview() {
    MyCustomTextComponent( displayString: "KotlinConf 2023")
}

@Composable
fun MyCustomTextComponent(displayString: String) {...}
```

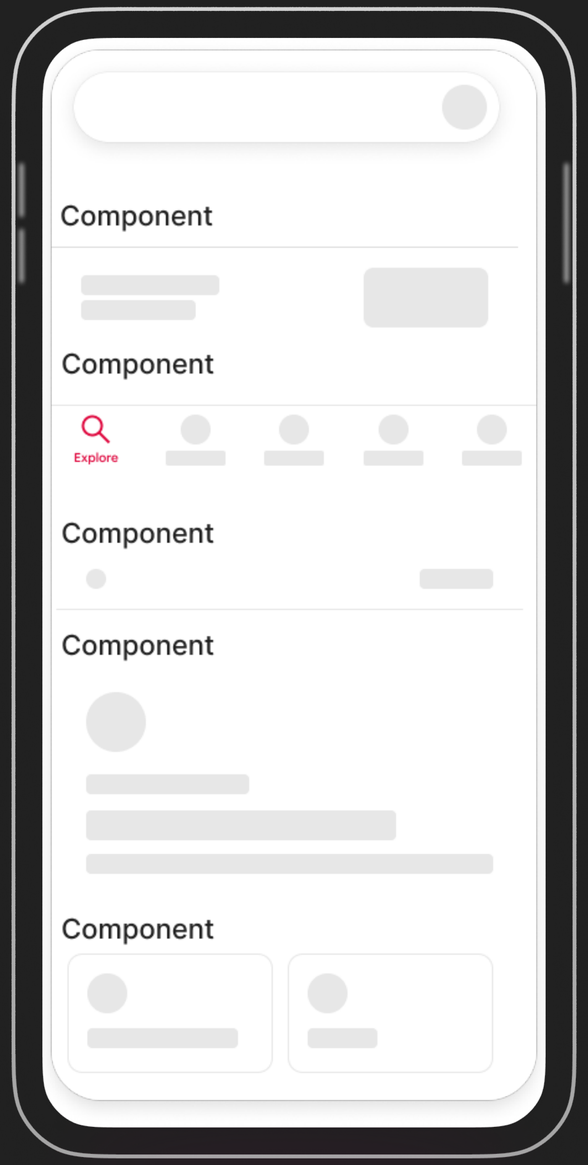
Revised Problem Statement:

Collect all functions annotated with @Preview and show them in a component browser

Kotlin Symbol Processing (KSP)

https://kotlinlang.org/docs/ksp-overview.html

```
KSFile
  packageName: KSName
  fileName: String
  annotations: List<KSAnnotation>  (File annotations)
  declarations: List<KSDeclaration>
    KSClassDeclaration // class, interface, object
      simpleName: KSName
      qualifiedName: KSName
      containingFile: String
      typeParameters: KSTypeParameter
      parentDeclaration: KSDeclaration
      classKind: ClassKind
      primaryConstructor: KSFunctionDeclaration
      superTypes: List<KSTypeReference>
      // contains inner classes, member functions, properties, etc.
      declarations: List<KSDeclaration>
    KSFunctionDeclaration // top level function
      simpleName: KSName
      qualifiedName: KSName
      containingFile: String
      typeParameters: KSTypeParameter
      parentDeclaration: KSDeclaration
      functionKind: FunctionKind
      extensionReceiver: KSTypeReference?
      returnType: KSTypeReference
      parameters: List<KSValueParameter>
      // contains local classes, local functions, local variables, etc.
      declarations: List<KSDeclaration>
```


Component
Component
Explore
Component
Component
Component

Component Name

Component Group

@Preview Function

```
data class ShowkaseBrowserComponent(
    val name: String,
    val group: String,
    val component: @Composable () -> Unit
)
```

```
data class ShowkaseBrowserComponent(
    val name: String,
    val group: String,
    val component: @Composable () -> Unit
)

interface ShowkaseProvider {
    fun componentList() : List<ShowkaseBrowserComponent>
}
```

```
class GeneratedCode: ShowkaseProvider {
    override fun componentList(): List<ShowkaseBrowserComponent> {
        return listOf(
            ShowkaseBrowserComponent(...),
            ShowkaseBrowserComponent(...),
            …
            ShowkaseBrowserComponent(...),
        )
    }
}
```

```
class GeneratedCode: ShowkaseProvider {
    override fun componentList(): List<ShowkaseBrowserComponent> {
        return listOf(
            ShowkaseBrowserComponent(...),
            ShowkaseBrowserComponent(...),
            …
            ShowkaseBrowserComponent(...),
        )
    }
}
```

```
class GeneratedCode: ShowkaseProvider {
    override fun componentList(): List<ShowkaseBrowserComponent> {
        return listOf(
            ShowkaseBrowserComponent(...),
            ShowkaseBrowserComponent(...),
            ...
            ShowkaseBrowserComponent(...),
        )
    }
}
```

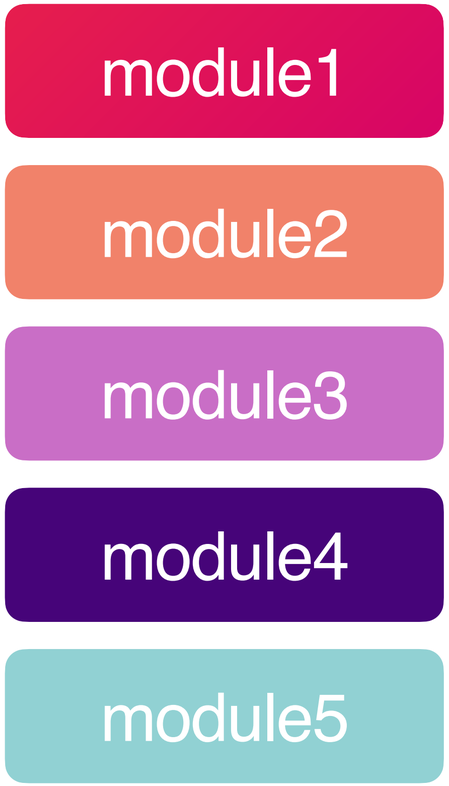
Which module has all the dependencies?

```
@Retention(AnnotationRetention.SOURCE)
@Target(AnnotationTarget.CLASS)
annotation class ShowkaseRoot
```

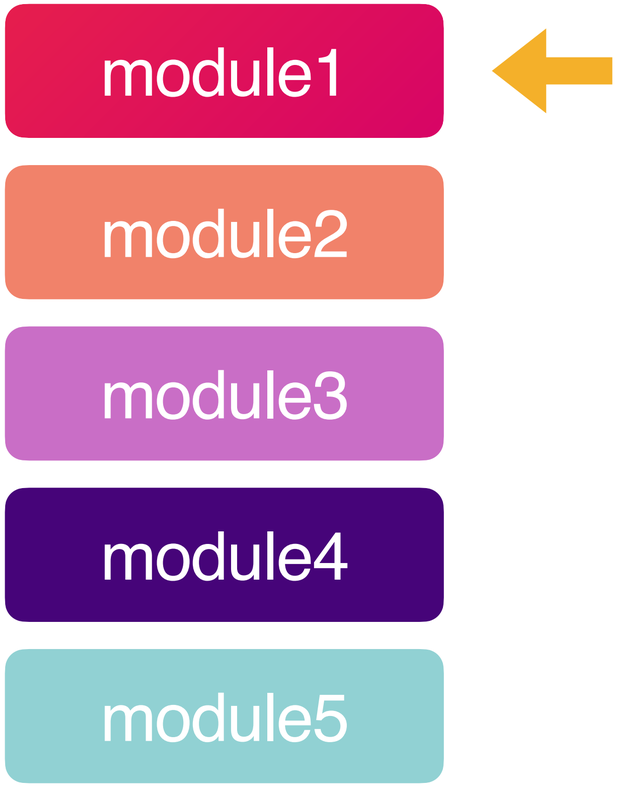
```
@Retention(AnnotationRetention.SOURCE)
@Target(AnnotationTarget.CLASS)
annotation class ShowkaseRoot


// Setup needed from the user of this library
@ShowkaseRoot
class MyShowkaseRoot
```

Which module has all the dependencies?

How can the root module access previews from other modules?

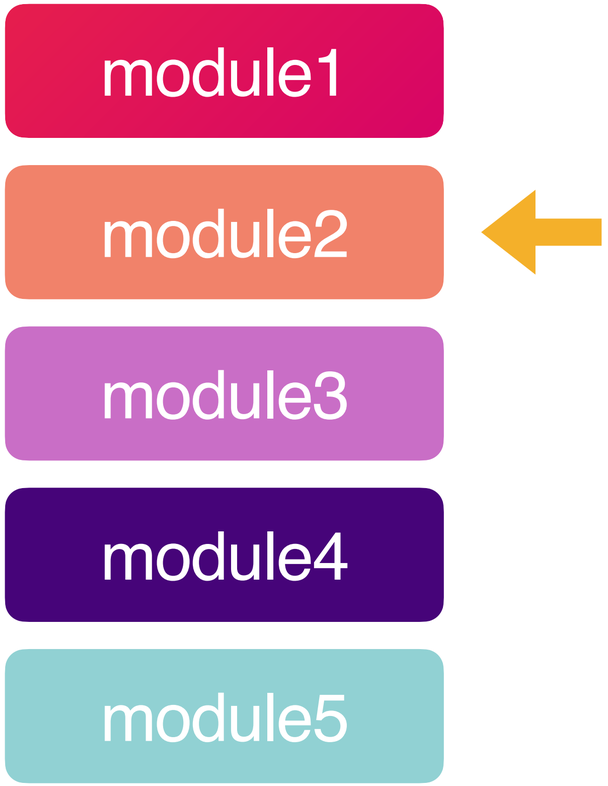


> Task :module1:kspDebugKotlin

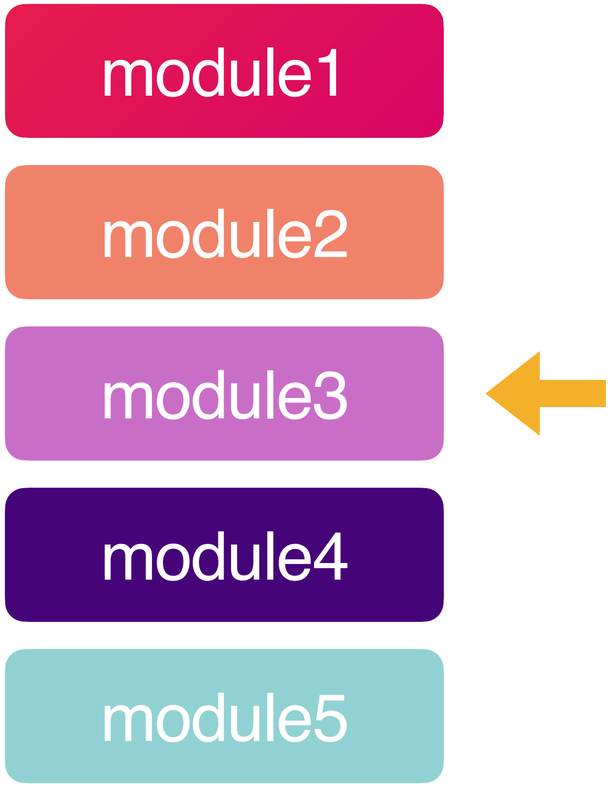


> Task :module1:kspDebugKotlin

> Task :module2:kspDebugKotlin




```
> Task :module1:kspDebugKotlin
> Task :module2:kspDebugKotlin
> Task :module3:kspDebugKotlin
```




```
> Task :module1:kspDebugKotlin
> Task :module2:kspDebugKotlin
> Task :module3:kspDebugKotlin
```


```
> Task :module1:kspDebugKotlin
> Task :module2:kspDebugKotlin
> Task :module3:kspDebugKotlin
```

module1

module2

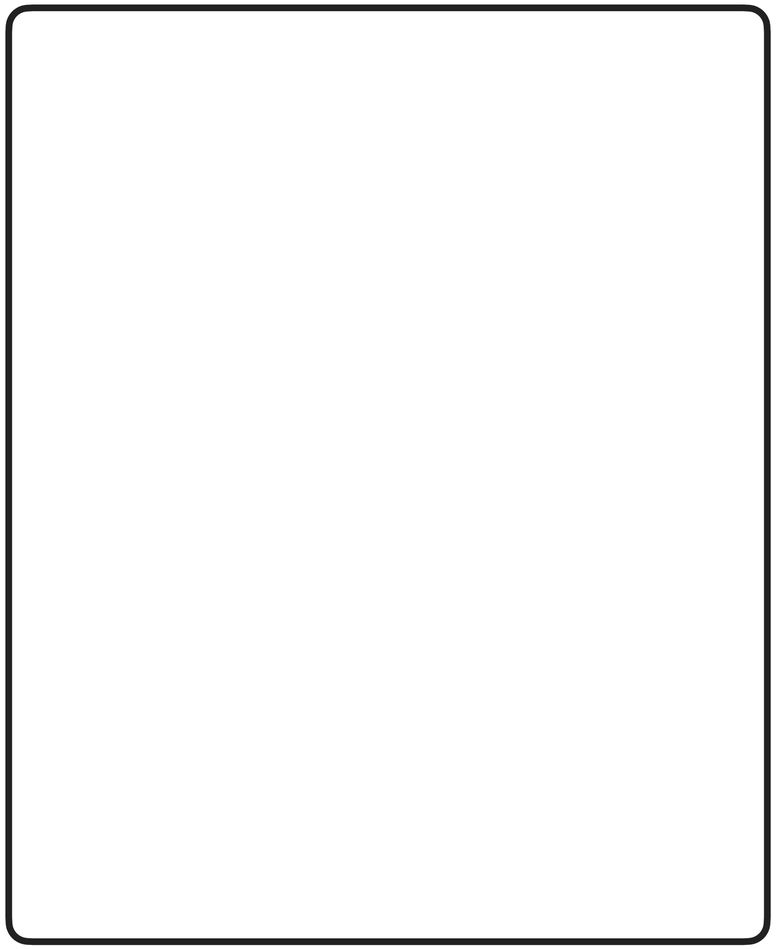
module3

module4

module5

Fixed package location

`com.airbnb.android.showkase`



module1

module2

module3

module4

module5

Fixed package location

`com.airbnb.android.showkase`

Metadata of previews from module 1

module1

module2

module3

module4

module5

Fixed package location

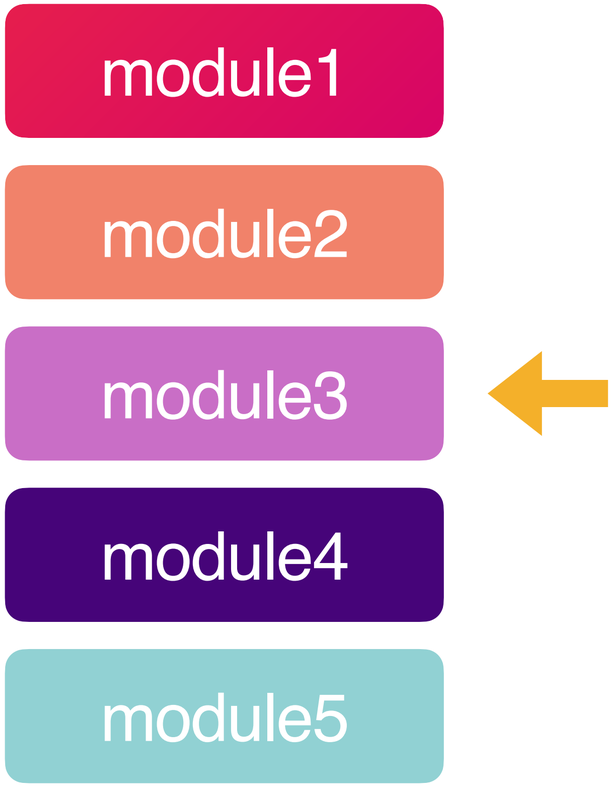
`com.airbnb.android.showkase`

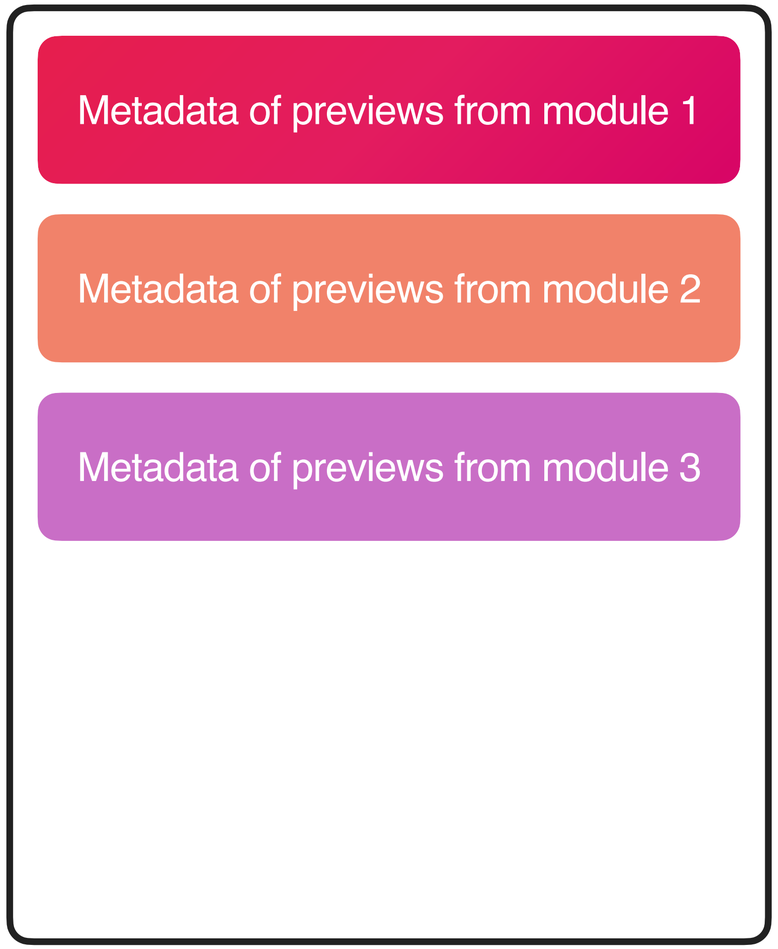
Metadata of previews from module 1

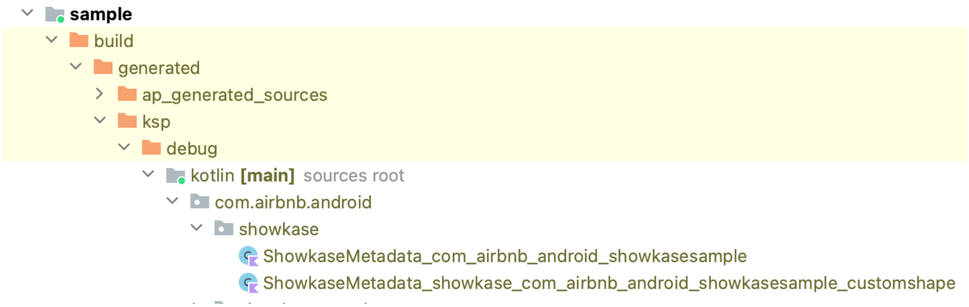
Metadata of previews from module 2

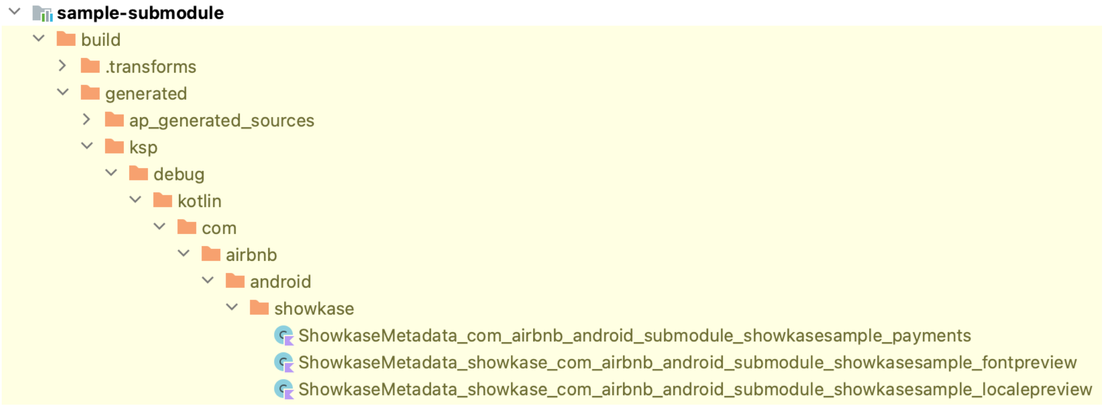
Fixed package location

`com.airbnb.android.showkase`

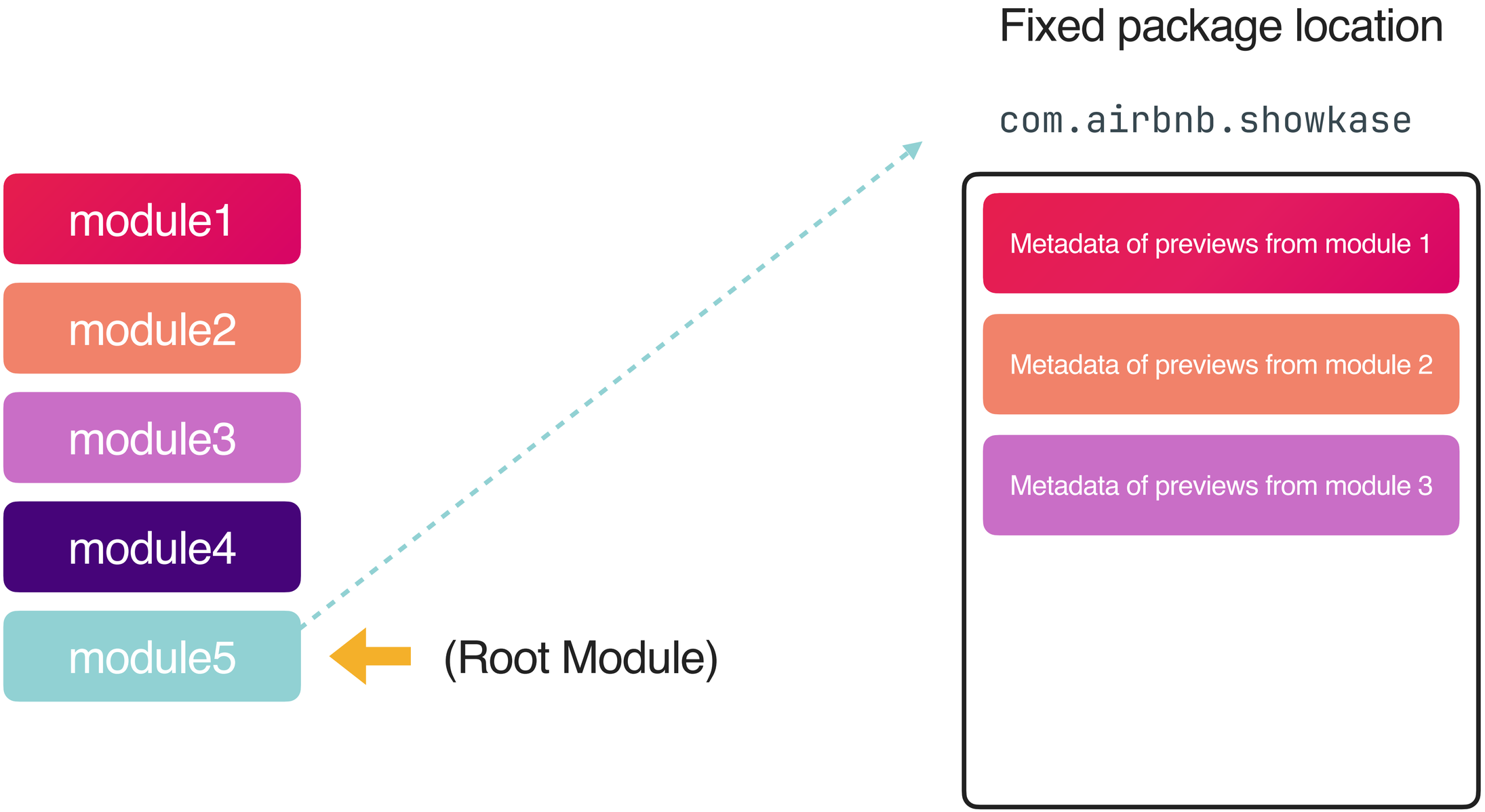






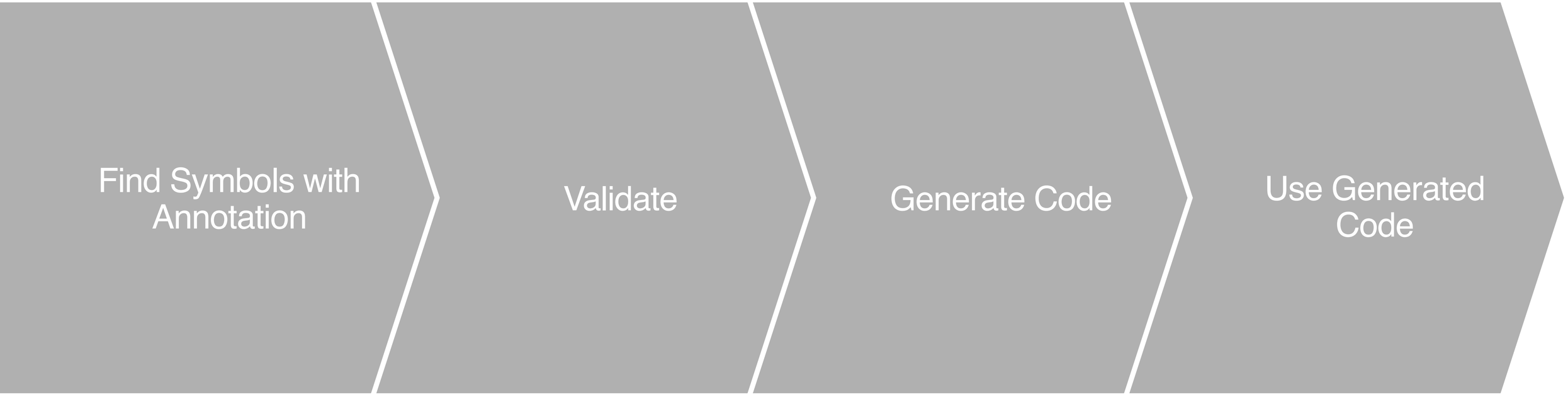


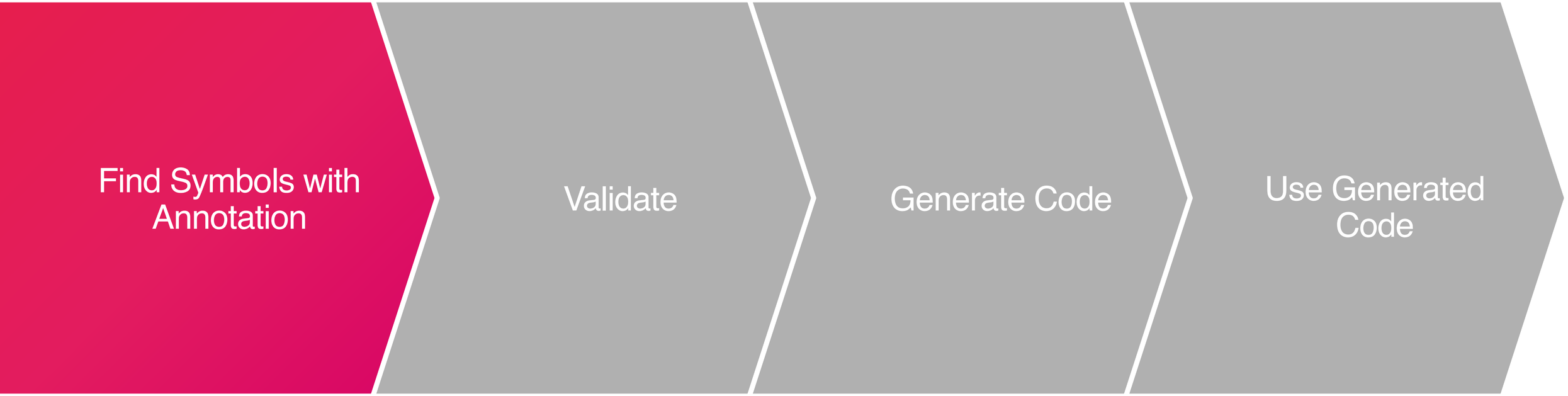
com.airbnb.android.showkase


Fixed package location
com.airbnb.showkase
module1
module2
module3
module4
module5
(Root Module)
Metadata of previews from module 1
Metadata of previews from module 2
Metadata of previews from module 3

Writing the Annotation Processor

```
class ShowkaseProcessor(
    val environment: SymbolProcessorEnvironment
) : SymbolProcessor {
     override fun process(resolver: Resolver): List<KSAnnotated> {

         return emptyList()
     }
}
```


Find Symbols with Annotation
Validate
Generate Code
Use Generated Code


Find Symbols with Annotation
Validate
Generate Code
Use Generated Code

```
class ShowkaseProcessor(
    val environment: SymbolProcessorEnvironment
) : SymbolProcessor {
     override fun process(resolver: Resolver): List<KSAnnotated> {
         // TODO: Step1 - Find elements with annotation

         return emptyList()
     }
}
```

```
// Find all preview functions in the current module
resolver.getSymbolsWithAnnotation("androidx.compose.ui.tooling.preview.Preview")
```

```
resolver.getSymbolsWithAnnotation("androidx.compose.ui.tooling.preview.Preview")
    .map { function ->

    }
```

```
resolver.getSymbolsWithAnnotation(...)
    .map { function →

    }
```

```
@Preview(
    name = "ImageRow",
    group = "Rows"
)
@Composable
fun ImageRowPreview() {
    ImageRow(...)
}
```

```
resolver.getSymbolsWithAnnotation(...)
    .map { function →

    }
```

```
@Preview(...)
@Composable
fun ImageRowPreview() {
    ImageRow(...)
}

object MyPreviewsObject {
    @Preview(...)
    @Composable
    fun ImageRowPreview() {
        ImageRow(...)
    }
}

class MyClass {
    @Preview(...)
    @Composable
    fun ImageRowPreview() {
        ImageRow(...)
    }
}
```

```
resolver.getSymbolsWithAnnotation("androidx.compose.ui.tooling.preview.Preview")
    .map { function ->
        val previewAnnotation = function.annotations.first {
            it.shortName.asString() == "Preview"
        }
    }
```

```
resolver.getSymbolsWithAnnotation("androidx.compose.ui.tooling.preview.Preview")
    .map { function ->
        val previewAnnotation = function.annotations.first {
            it.shortName.asString() == "Preview"
        }
        val name = previewAnnotation.arguments.first {
            it.name.asString() == "name"
        }.value
        val group = previewAnnotation.arguments.first {
            it.name.asString() == "group"
        }.value
    }
```

```
resolver.getSymbolsWithAnnotation("androidx.compose.ui.tooling.preview.Preview")
    .map { function →
        val previewAnnotation = function.annotations.first {
            it.shortName.asString() == "Preview"
        }
        val name = previewAnnotation.arguments.first {
            it.name.asString() == "name"
        }.value
        val group = previewAnnotation.arguments.first {
            it.name.asString() == "group"
        }.value
      val wrapperClassMetadata = if (function.parentDeclaration is KSClassDeclaration){
        val wrappingClass = (function.parentDeclaration as KSClassDeclaration)
        wrappingClass.simpleName to wrappingClass.classKind
      } else null to null
    }
```

MyClass

ClassKind.Class or ClassKind.Object

```
resolver.getSymbolsWithAnnotation("androidx.compose.ui.tooling.preview.Preview")
    .map { function ->
        ...
        Metadata(
            symbol = function,
            previewName = name,
            previewGroup = group,
            wrapperDeclarationName = wrapperClassMetadata.first,
            classKind = wrapperClassMetadata.second
        )
    }
```

```kotlin
@Retention(AnnotationRetention.SOURCE)
@Target(AnnotationTarget.FUNCTION)
annotation class ShowkaseComposable(
    val name: String = "",
    val group: String = "",
)
```

```kotlin
@ShowkaseComposable(name = "MyCustomComponent", group = "Custom")
@Preview(name = "MyCustomComponent", group = "SectionHeader")
@Composables
fun MyCustomComponentPreview() {
    MyCustomComponent("KotlinConf 2023")
}
```

```
val previewList: List<Metadata> = processPreviewFunctions(...)
val showkasePreviewList: List<Metadata> = processPreviewFunctions(...)
```

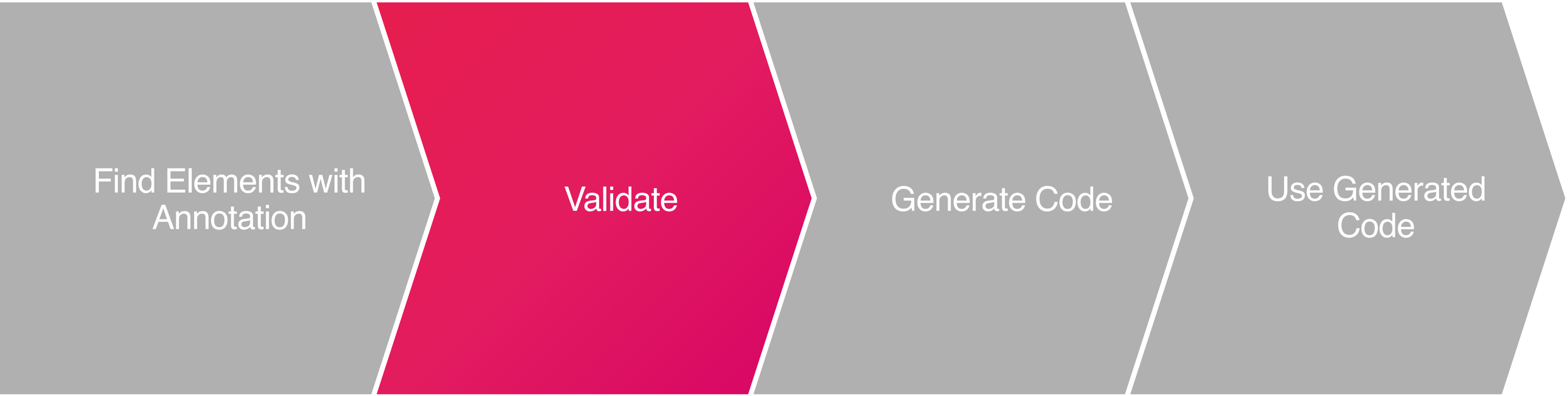
```
val previewList: List<Metadata> = processPreviewFunctions(...)
val showkasePreviewList: List<Metadata> = processPreviewFunctions(...)

val consolidatedPreviewList = (showkasePreviewList + previewList)
```

```
val previewList: List<Metadata> = processPreviewFunctions(...)
val showkasePreviewList: List<Metadata> = processPreviewFunctions(...)

val consolidatedPreviewList = (showkasePreviewList + previewList)
    .distinctBy {
        it.declaration.packageName + it.wrapperDeclarationName +
            it.declaration.simpleName
    }
```

```
val previewList: List<Metadata> = processPreviewFunctions(...)
val showkasePreviewList: List<Metadata> = processPreviewFunctions(...)

val consolidatedPreviewList = (showkasePreviewList + previewList)
    .distinctBy {
        it.declaration.packageName + it.wrapperDeclarationName +
            it.declaration.simpleName
    }
    .distinctBy {
        it.previewGroup + it.previewName
    }
```

```
// Check if the current module is the root module
val rootModule = resolver.getSymbolsWithAnnotation(
    ShowkaseRoot::class.java.name
).firstOrNull()
```

```
val previewList: List<Metadata> = processPreviewFunctions(...)
val showkasePreviewList: List<Metadata> = processPreviewFunctions(...)
val consolidatedPreviewList = (showkasePreviewList + previewList)
    .distinctBy {
        it.declaration.packageName + it.wrapperDeclarationName +
            it.declaration.simpleName
    }
    .distinctBy {
        it.previewGroup + it.previewName
    }

val rootModule: KSAnnotated? = processRootAnnotation(...)
```

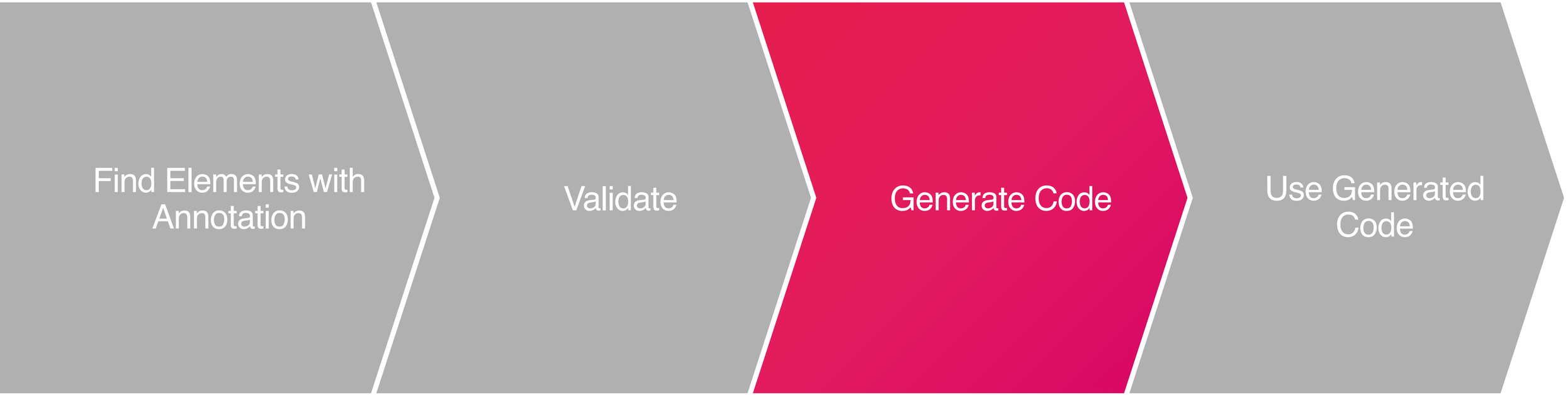

Find Elements with Annotation
Validate
Generate Code
Use Generated Code

Validation

🕵🏻 Preview functions aren’t private

👌🏻 Preview functions have no non-default parameters

🧩 Only @Composable functions can be annotated with @ShowkaseComposable

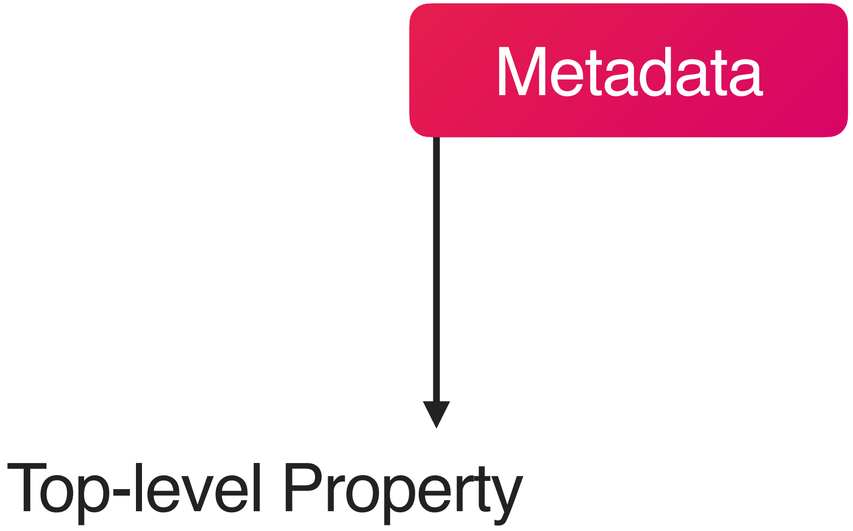

Find Elements with Annotation
Validate
Generate Code
Use Generated Code

Square/KotlinPoet

https://github.com/square/kotlinpoet

```
val consolidatePreviewList: List<Metadata>
```

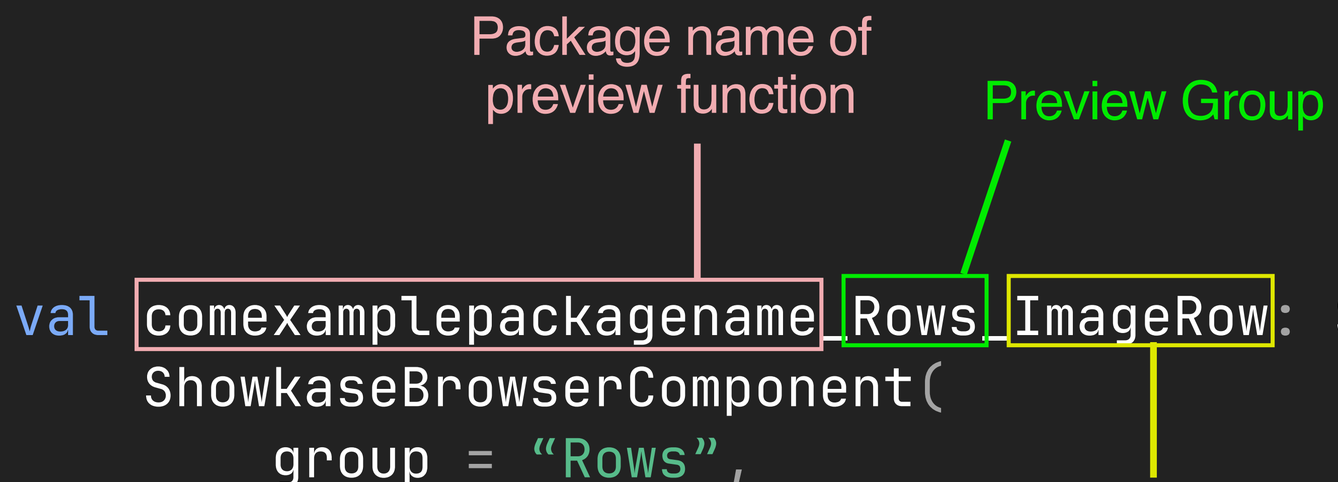
Previews from current module that’s being processed by KSP


Metadata
Top-level Property

Top-level Property

```
val comexamplepackagename_Rows_ImageRow: ShowkaseBrowserComponent =
    ShowkaseBrowserComponent(
        group = “Rows”,
        name = “ImageRow”,
        composable = @Composable {
            ImageRowPreview()
        }
    )
```

Top-level Property




```
val comexamplepackagename_Rows_ImageRow: ShowkaseBrowserComponent =
    ShowkaseBrowserComponent(
        group = “Rows”,
        name = “ImageRow”,
        composable = @Composable {
            ImageRowPreview()
        }
    )
```

Top-level Property

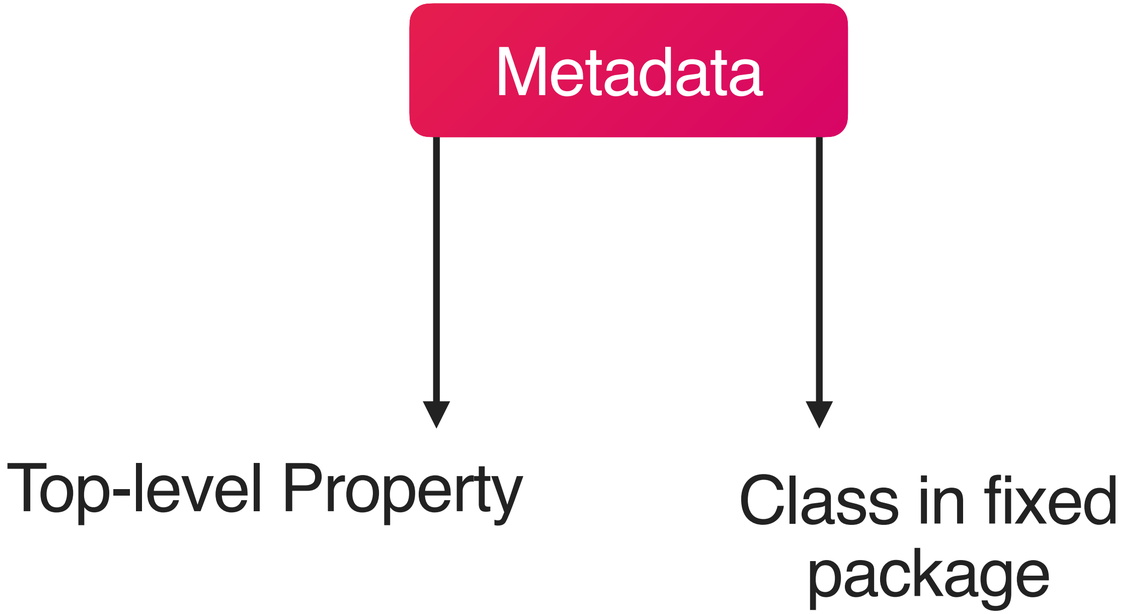
```
val comexamplepackagename_Rows_ImageRow: ShowkaseBrowserComponent =
    ShowkaseBrowserComponent(
        group = “Rows”,
        name = “ImageRow”,
        composable = @Composable {
            ImageRowPreview()
        }
    )
```

Top-level Property

```
val comexamplepackagename_Rows_ImageRow: ShowkaseBrowserComponent =
    ShowkaseBrowserComponent(
        group = “Rows”,
        name = “ImageRow”,
        composable = @Composable {
            MyClass().ImageRowPreview()
        }
    )
```

Top-level Property

```
val comexamplepackagename_Rows_ImageRow: ShowkaseBrowserComponent =
    ShowkaseBrowserComponent(
        group = “Rows”,
        name = “ImageRow”,
        composable = @Composable {
            MyObject.ImageRowPreview()
        }
    )
```


Metadata
Top-level Property
Class in fixed package

Class in fixed package

```
package com.airbnb.android.showkase

@ShowkaseCodegenMetadata(
  packageName = “com.example.packagename”,
  generatedPropertyName = “comexamplepackagename_Rows_ImageRow”,
)
class ShowkaseMetadata_comexamplepackagename_Rows_ImageRow {}
```

Class in fixed package

```
package com.airbnb.android.showkase

@ShowkaseCodegenMetadata(
  packageName = “com.example.packagename”,
  generatedPropertyName = “comexamplepackagename_Rows_ImageRow”,
)
class ShowkaseMetadata_comexamplepackagename_Rows_ImageRow {}
```

Class in fixed package

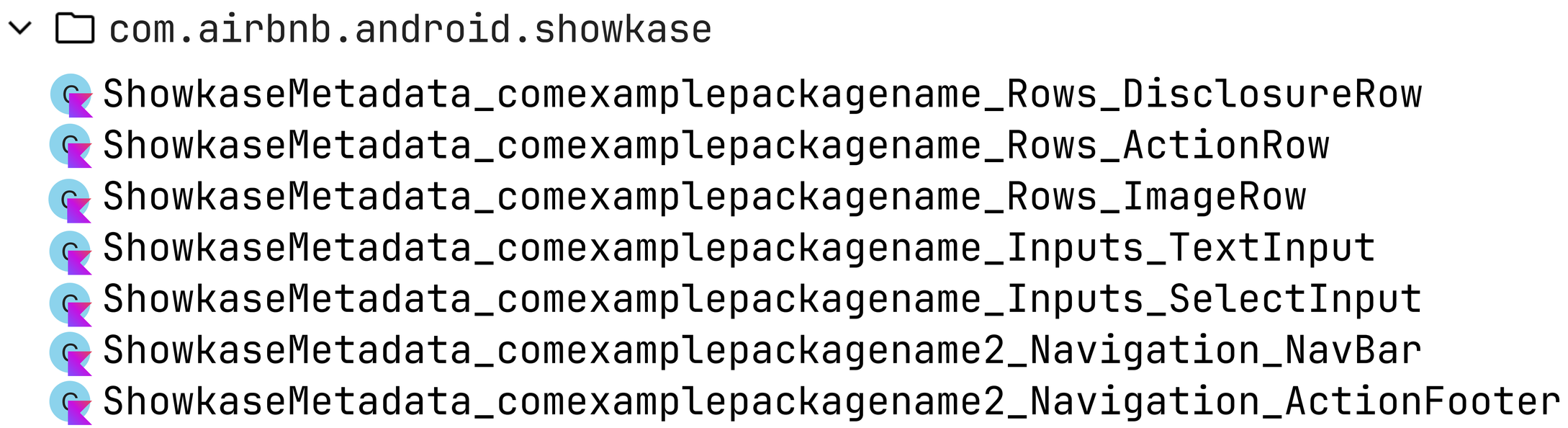
```
package com.airbnb.android.showkase

@ShowkaseCodegenMetadata(
  packageName = “com.example.packagename”,
  generatedPropertyName = “comexamplepackagename_Rows_ImageRow”,
)
class ShowkaseMetadata_comexamplepackagename_Rows_ImageRow {}
```

Class in fixed package

```
package com.airbnb.android.showkase

@ShowkaseCodegenMetadata(
  packageName = “com.example.packagename”,
  generatedPropertyName = “comexamplepackagename_Rows_ImageRow”,
)
class ShowkaseMetadata_comexamplepackagename_Rows_ImageRow {}
```

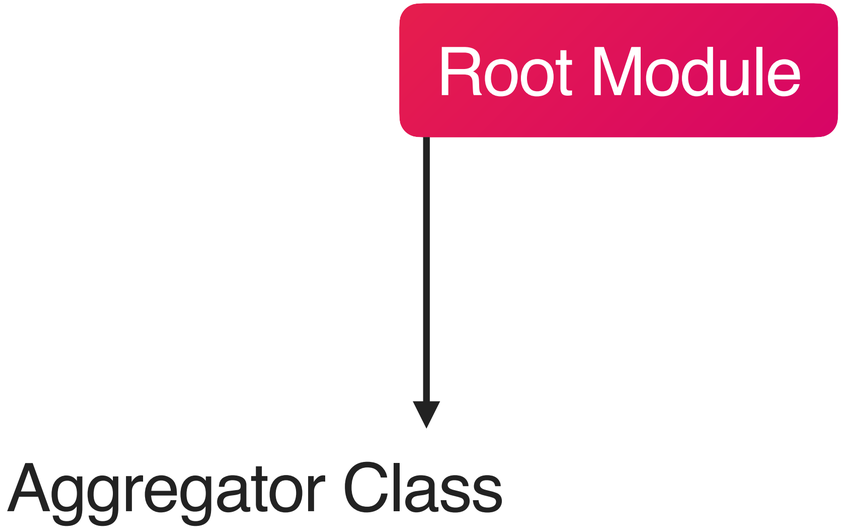
Class in fixed package

```
package com.airbnb.android.showkase

@ShowkaseCodegenMetadata(
  packageName = “com.example.packagename”,
  generatedPropertyName = “comexamplepackagename_Rows_ImageRow”,
)
class ShowkaseMetadata_comexamplepackagename_Rows_ImageRow {}
```


com.airbnb.android.showkase
ShowkaseMetadata_comexamplepackagename_Rows_DisclosureRow
ShowkaseMetadata_comexamplepackagename_Rows_ActionRow
ShowkaseMetadata_comexamplepackagename_Rows_ImageRow
ShowkaseMetadata_comexamplepackagename_Inputs_TextInput
ShowkaseMetadata_comexamplepackagename_Inputs_SelectInput
ShowkaseMetadata_comexamplepackagename2_Navigation_NavBar
ShowkaseMetadata_comexamplepackagename2_Navigation_ActionFooter

```
val rootModule: KSAnnotated? = processRootAnnotation(…)
```

```
val rootModule: KSAnnotated? = processRootAnnotation(…)
if (rootModule == null) {
    // This means that we are currently processing
    // the root module
}
```


Root Module
Aggregator Class

Aggregator Class

```
resolver.getDeclarationsFromPackage(“com.airbnb.android.showkase")
```

Aggregator Class

```
resolver.getDeclarationsFromPackage(“com.airbnb.android.showkase")
    .filter { it is KSClassDeclaration }
```

Aggregator Class

```
resolver.getDeclarationsFromPackage(“com.airbnb.android.showkase")
    .filter { it is KSClassDeclaration }
    .map {

    }
```

Aggregator Class

```
resolver.getDeclarationsFromPackage(“com.airbnb.android.showkase")
    .filter { it is KSClassDeclaration }
    .map {
        val annotations = it.getAnnotationsByType(
            ShowkaseCodegenMetadata::class
        )
    }
```

Aggregator Class

```
resolver.getDeclarationsFromPackage(...)
    .filter { it is KSClassDeclaration }
    .map {
        val annotations = it.getAnnotationsByType(
            ShowkaseCodegenMetadata::class
        )
        annotations.firstOrNull()?.let { annotation ->
           GeneratedMetadata(
                propertyName = annotation.generatedPropertyName,
                propertyPackage = annotation.packageName,
                classKind = ClassKind.CLASS,
                originatingNode = it
            )
        }
    }
```

Aggregator Class

```
resolver.getDeclarationsFromPackage(...)
    .filter { it is KSClassDeclaration }
    .map {
        val annotations = it.getAnnotationsByType(
            ShowkaseCodegenMetadata::class
        )
        annotations.firstOrNull()?.let { annotation ->
            GeneratedMetadata(
                propertyName = annotation.generatedPropertyName,
                propertyPackage = annotation.packageName,
                classKind = ClassKind.CLASS,
                originatingNode = it
            )
        }

    }
    .filterNotNull()
    .toList()
```

Aggregator Class

```
val rootModule: KSAnnotated? = processRootAnnotation(...)
if (rootModule != null) {

}
```

Aggregator Class

```
val rootModule: KSAnnotated? = processRootAnnotation(...)
if (rootModule != null) {
    val fixedPackageMetadataList = getFixedPackageMetadata(...)
}
```

Aggregator Class

```
val rootModule: KSAnnotated? = processRootAnnotation(...)
if (rootModule != null) {
    val fixedPackageMetadataList = getFixedPackageMetadata(...)
    val aggregatedMetadata = fixedPackageMetadataList +
        currentPreviewList.map { it.toGeneratedMetadata() }
}
```

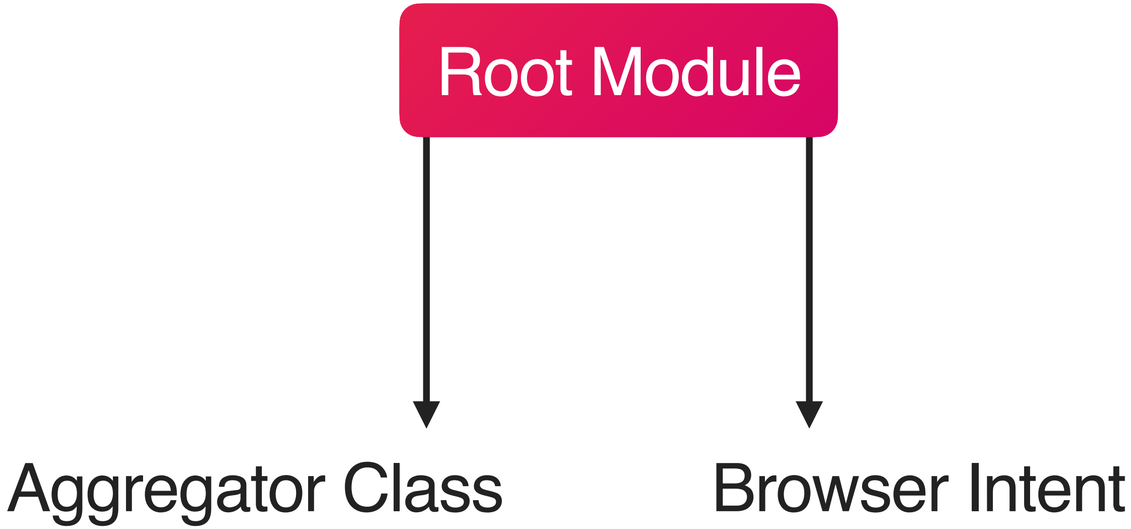
Aggregator Class

```
val rootModule: KSAnnotated? = processRootAnnotation(...)
if (rootModule ≠ null) {
    val fixedPackageMetadataList = getFixedPackageMetadata(...)
    val aggregatedMetadata = fixedPackageMetadataList +
        currentPreviewList.map { it.toGeneratedMetadata() }
    writeAggregatedFile(aggregatedMetadata)
}
```

Aggregator Class

```
class MyShowkaseRoot_ShowkaseCodegen : ShowkaseProvider {
  override fun getShowkaseComponents() = listOf(
    comexamplepackagename_Rows_DisclosureRow,
    comexamplepackagename_Rows_ActionRow,
    comexamplepackagename_Rows_ImageRow,
    comexamplepackagename_Inputs_TextInput,
    comexamplepackagename_Inputs_SelectInput,
    comexamplepackagename2_Navigation_NavBar,
    comexamplepackagename2_Navigation_ActionFooter,
  )
}
```

```
val comexamplepackagename_Rows_ImageRow = ShowkaseBrowserComponent(
    group = "Rows",
    name = "ImageRow",
    composable = @Composable {
        ImageRowPreview()
    }
)
```


Root Module
Aggregator Class
Browser Intent

Browser Intent

```
object Showkase
```

Browser Intent

```
fun Showkase.getBrowserIntent(context: Context): Intent {

}
```

Browser Intent

```
fun Showkase.getBrowserIntent(context: Context): Intent {
    val intent = Intent(context, ShowkaseBrowserActivity::class.java)
}
```

Browser Intent

```
fun Showkase.getBrowserIntent(context: Context): Intent {
    val intent = Intent(context, ShowkaseBrowserActivity::class.java)
    intent.putExtra(“SHOWKASE_AGGREGATOR_FILE”,
      “com.yourpackage.MyRootModule_ShowkaseCodegen”)
}
```

Browser Intent

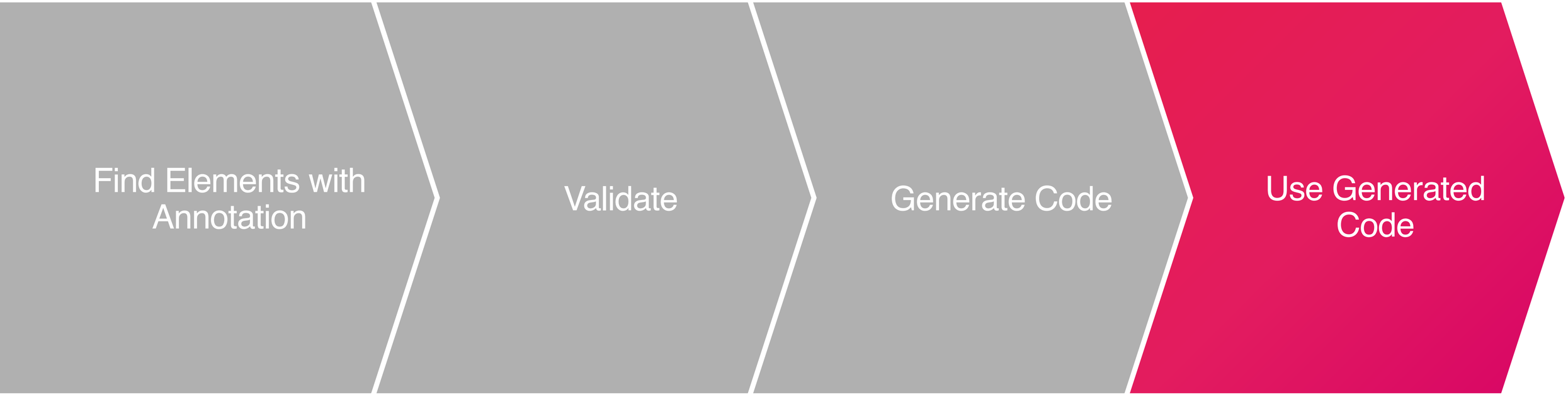
```
fun Showkase.getBrowserIntent(context: Context): Intent {
    val intent = Intent(context, ShowkaseBrowserActivity::class.java)
    intent.putExtra(“SHOWKASE_AGGREGATOR_FILE”,
      “com.yourpackage.MyRootModule_ShowkaseCodegen”)
    return intent
}
```

Browser Intent

```
fun Showkase.getBrowserIntent(context: Context): Intent {
    val intent = Intent(context, ShowkaseBrowserActivity::class.java)
    intent.putExtra(“SHOWKASE_AGGREGATOR_FILE”,
      “com.yourpackage.MyRootModule_ShowkaseCodegen”)
    return intent
}
```

Browser Intent

```
fun Showkase.getBrowserIntent(context: Context): Intent {
    val intent = Intent(context, ShowkaseBrowserActivity::class.java)
    intent.putExtra(“SHOWKASE_AGGREGATOR_FILE”,
“com.yourpackage.MyRootModule_ShowkaseCodegen”)
    return intent
}

// Usage
startActivity(Showkase.getBrowserIntent(requireContext()))
```


Find Elements with Annotation
Validate
Generate Code
Use Generated Code

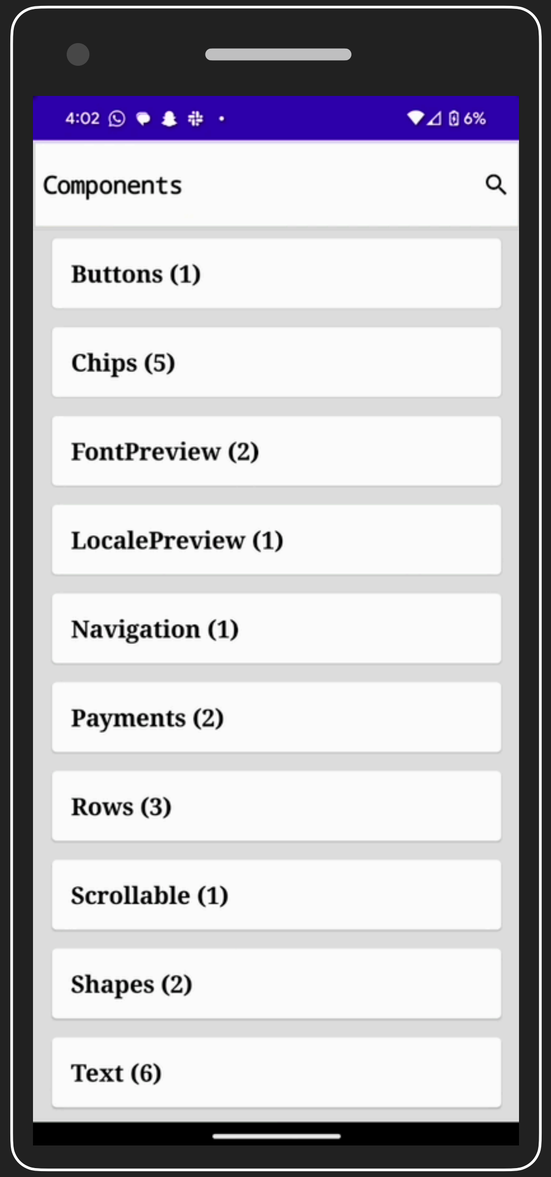
```
class ShowkaseBrowserActivity: ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {

    }
}
```

```
class ShowkaseBrowserActivity: ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        val classKey = intent.extras.getString(“SHOWKASE_AGGREGATOR_FILE”)
    }
}
```

```
class ShowkaseBrowserActivity: ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        val classKey = intent.extras.getString(“SHOWKASE_AGGREGATOR_FILE”)
        val provider = Class.forName(classKey)
            .newInstance()
    }
}
```

```
class ShowkaseBrowserActivity: ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        val classKey = intent.extras.getString(“SHOWKASE_AGGREGATOR_FILE”)
        val provider = Class.forName(classKey)
            .newInstance()
        val showkaseMetadata = (provider as ShowkaseProvider)
            .componentList()
    }
}
```

```
class ShowkaseBrowserActivity: ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        val classKey = intent.extras.getString(“SHOWKASE_AGGREGATOR_FILE”)
        val provider = Class.forName(classKey)
            .newInstance()
        val showkaseMetadata = (provider as ShowkaseProvider)
            .componentList()

        // Regular app from this point onwards
    }
}
```


4:02
6%
Components
Buttons (1)
Chips (5)
FontPreview (2)
LocalePreview (1)
Navigation (1)
Payments (2)
Rows (3)
Scrollable (1)
Shapes (2)
Text (6)

Product Card [Basic Example]

Title

Lorem ipsum dolor sit

Basic Example

Product Card [Dark Mode]

Title

Lorem ipsum dolor sit

Dark mode enabled

Product Card [RTL]

Title

Lorem ipsum dolor sit

Right-to-left enabled

Product Card [Font Scaled x 2]

Title

Lorem ipsum dolor sit

Font size is scaled to 2X the normal size

Product Card [Display Scaled x 2]

Title

Lorem ipsum dolor sit

Density is scaled to 2X the normal size

Button

Button

Button

Button

Button

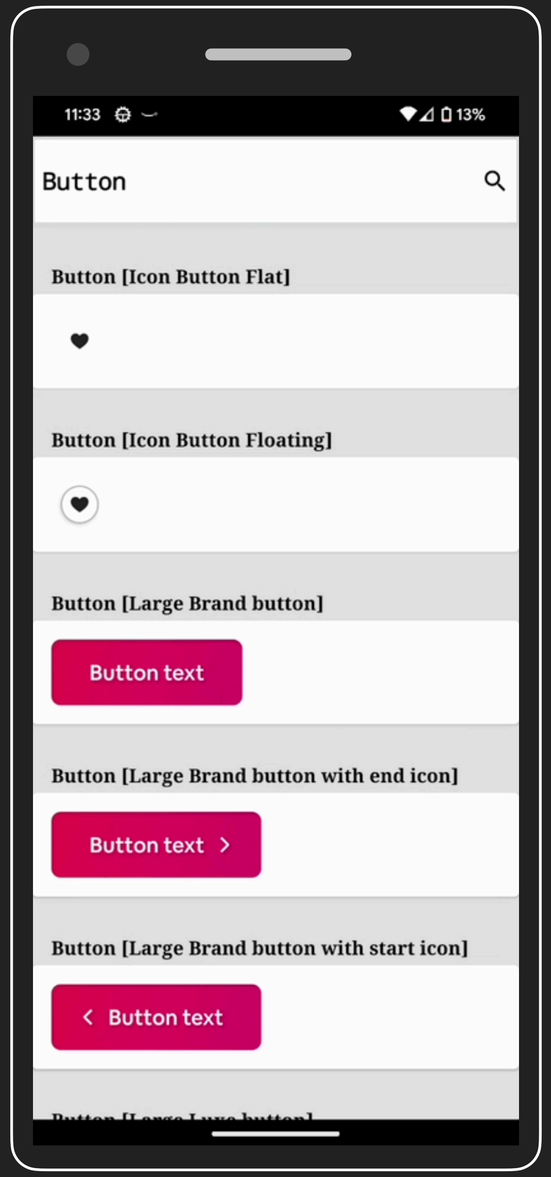
I would like to link the style variants of a component together 💅

```kotlin
@Retention(AnnotationRetention.SOURCE)
@Target(AnnotationTarget.FUNCTION)
annotation class ShowkaseComposable(
    val name: String = "",
    val group: String = "",
)
```

```kotlin
@Retention(AnnotationRetention.SOURCE)
@Target(AnnotationTarget.FUNCTION)
annotation class ShowkaseComposable(
    val name: String = "",
    val group: String = "",
    val styleName: String = "",
    val defaultStyle: Boolean = false
)
```

```
@ShowkaseComposable(name = "Button", group = “Inputs”, defaultStyle = true)
@Composable
fun Preview_Button_Default() {
    ...
}
```

```
@ShowkaseComposable(name = "Button", defaultStyle = true)
@Composable
fun Preview_Button_Default() {
    ...
}

@ShowkaseComposable(name = "Button", group = “Inputs”, styleName = "Plus")
@Composable
fun Preview_Button_PlusStyle() {
    ...
}

@ShowkaseComposable(name = "Button", group = “Inputs”, styleName = "Lux")
@Composable
fun Preview_Button_LuxStyle() {
    ...
}
```


11:33
13%
Button
Button [Icon Button Flat]
Button [Icon Button Floating]
Button [Large Brand button]
Button text
Button [Large Brand button with end icon]
Button text
Button [Large Brand button with start icon]
Button text

I would like to document
other UI Elements
as well 🌈

```kotlin
@Retention(AnnotationRetention.SOURCE)
@Target(AnnotationTarget.FIELD)
annotation class ShowkaseTypography(
    val name: String = "",
    val group: String = "",
)

@Retention(AnnotationRetention.SOURCE)
@Target(AnnotationTarget.FIELD)
annotation class ShowkaseColor(
    val name: String = "",
    val group: String = "",
)
```

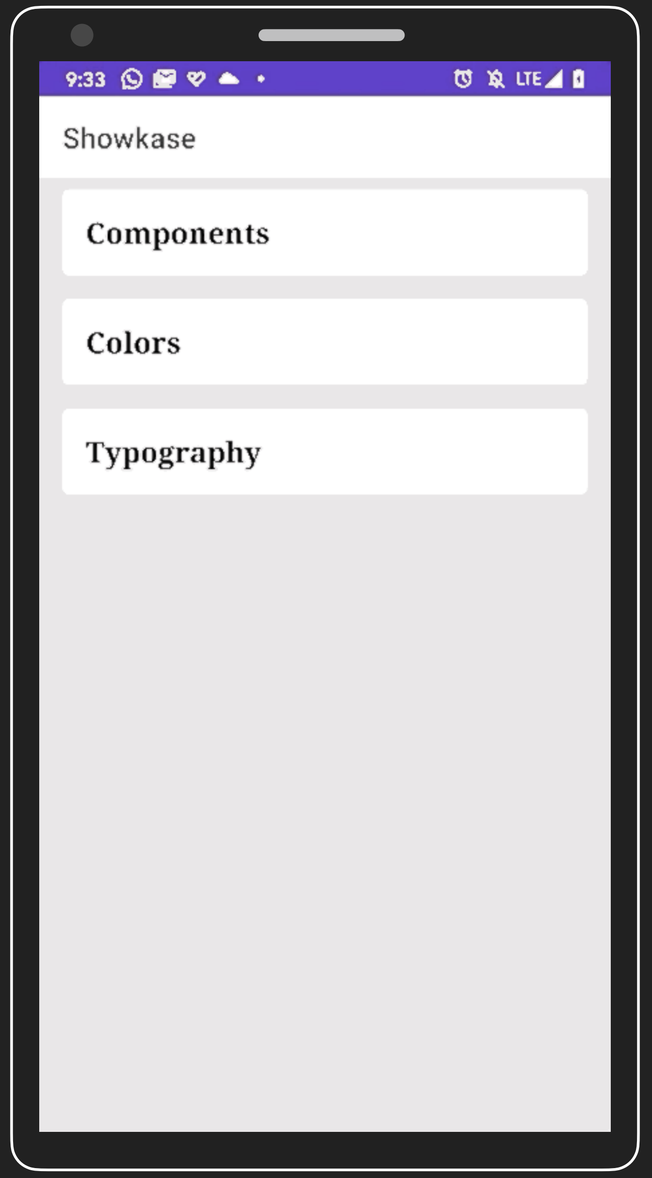
```kotlin
@ShowkaseColor(name = "Primary Color", group = "Material Design")
val primaryColor = Color(0xFF6200EE)

@ShowkaseTypography(name = "Heading1", group = "Material Design")
val h1 = TextStyle(
    fontWeight = FontWeight.Light,
    fontSize = 96.sp,
    letterSpacing = (-1.5).sp
)
```

```
val previewList: List<Metadata> = processPreviewFunctions(...)
val showkasePreviewList: List<Metadata> = processPreviewFunctions(...)
```

```
val previewList: List<Metadata> = processPreviewFunctions(...)
val showkasePreviewList: List<Metadata> = processPreviewFunctions(...)

val colorList: List<Metadata> = processColors(...)
val typography: List<Metadata> = processTypography(...)
```

```
interface ShowkaseProvider {
    fun componentList(): List<BrowserComponent>
}
```

```
interface ShowkaseProvider {
    fun componentList(): List<BrowserComponent>

    fun colors(): List<BrowserColor>

    fun typography: List<BrowserTypography>
}
```


9:33
LTE
Showkase
Components
Colors
Typography

Quality

```
interface ShowkaseProvider {
    fun componentList(): List<BrowserComponent>

    fun colors(): List<BrowserColor>

    fun typography: List<BrowserTypography>
}
```

```
interface ShowkaseProvider {
    fun componentList(): List<BrowserComponent>

    fun colors(): List<BrowserColor>

    fun typography: List<BrowserTypography>

    fun metadata(): ShowkaseElementsMetadata {
        val componentList = componentList()
        val colorList = colors()
        val typographyList = typography()
        return ShowkaseElementsMetadata(componentList, colorList, typographyList)
    }
}
```

```
fun Showkase.getMetadata(): ShowkaseElementsMetadata {

}
```

```kotlin
fun Showkase.getMetadata(): ShowkaseElementsMetadata {
    try {
      val provider = Class.forName(
          "com.example.packagename.RootModule_ShowkaseCodegen"
        ).newInstance() as ShowkaseProvider
      return showkaseComponentProvider.metadata()
    } catch(exception: ClassNotFoundException) {
      error("Make sure that you have setup Showkase correctly...")
    }
}
```

```
val showkaseMetadata = Showkase.metadata()
```

```
val showkaseMetadata = Showkase.metadata()

val components = showkaseMetadata.componentList()
val colors = showkaseMetadata.colorList()
val typography = showkaseMetadata.typographyList()
```

Having access to all UI elements that you care about is really POWERFUL 💪

Use case:
Screenshot Testing

CashApp/Paparazzi

https://github.com/CashApp/Paparazzi

```kotlin
@RunWith(TestParameterInjector::class)
class MyPaparazziShowkaseScreenshotTest_PaparazziShowkaseTest :MyScreenshotTest() {
  @get:Rule
  val paparazzi: Paparazzi = providePaparazzi()

  @Test
  fun test_previews(
    @TestParameter(valuesProvider = PaparazziShowkasePreviewProvider::class)
        elementPreview: PaparazziShowkaseTestPreview,
    @TestParameter(valuesProvider = PaparazziShowkaseDeviceConfigProvider::class)
        config: PaparazziShowkaseDeviceConfig,
    @TestParameter(valuesProvider = PaparazziShowkaseLayoutDirectionProvider::class)
        direction: LayoutDirection,
    @TestParameter(valuesProvider = PaparazziShowkaseUIModeProvider::class)
        uiMode: PaparazziShowkaseUIMode,
  ): Unit {
    paparazzi.unsafeUpdateConfig(config.deviceConfig.copy(softButtons = false))
    takePaparazziSnapshot(paparazzi, elementPreview, direction, uiMode)
  }

  private object PaparazziShowkasePreviewProvider : TestParameter.TestParameterValuesProvider {
    override fun provideValues(): List<PaparazziShowkaseTestPreview> {
      val metadata = Showkase.getMetadata()
      val components = metadata.componentList.map(::ComponentTestPreview)
      val colors = metadata.colorList.map(::ColorTestPreview)
      val typography = metadata.typographyList.map(::TypographyTestPreview)
      return components + colors + typography
    }
  }

  private object PaparazziShowkaseDeviceConfigProvider : TestParameter.TestParameterValuesProvider {
    override fun provideValues(): List<PaparazziShowkaseDeviceConfig> = deviceConfigs()
  }

  private object PaparazziShowkaseLayoutDirectionProvider :
      TestParameter.TestParameterValuesProvider {
    override fun provideValues(): List<LayoutDirection> = layoutDirections()
  }

  private object PaparazziShowkaseUIModeProvider : TestParameter.TestParameterValuesProvider {
    override fun provideValues(): List<PaparazziShowkaseUIMode> = uiModes()
  }
}
```

```kotlin
@RunWith(TestParameterInjector::class)
class MyPaparazziShowkaseScreenshotTest_PaparazziShowkaseTest: MyScreenshotTest() {
  @get:Rule
  val paparazzi: Paparazzi = providePaparazzi()

  @Test
  fun test_previews(
    @TestParameter(valuesProvider = PaparazziShowkasePreviewProvider::class)
        elementPreview: PaparazziShowkaseTestPreview,
    @TestParameter(valuesProvider = PaparazziShowkaseDeviceConfigProvider::class)
        config: PaparazziShowkaseDeviceConfig,
    @TestParameter(valuesProvider = PaparazziShowkaseLayoutDirectionProvider::class)
        direction: LayoutDirection,
    @TestParameter(valuesProvider = PaparazziShowkaseUIModeProvider::class)
        uiMode: PaparazziShowkaseUIMode,
  ): Unit {
    paparazzi.unsafeUpdateConfig(config.deviceConfig.copy(softButtons = false))
    takePaparazziSnapshot(paparazzi, elementPreview, direction, uiMode)
  }
```

```
@RunWith(TestParameterInjector::class)
class MyPaparazziShowkaseScreenshotTest_PaparazziShowkaseTest: MyScreenshotTest() {
  @get:Rule
  val paparazzi: Paparazzi = providePaparazzi()

  @Test
  fun test_previews(
    @TestParameter(valuesProvider = PaparazziShowkasePreviewProvider::class)
        elementPreview: PaparazziShowkaseTestPreview,
    @TestParameter(valuesProvider = PaparazziShowkaseDeviceConfigProvider::class)
        config: PaparazziShowkaseDeviceConfig,
    @TestParameter(valuesProvider = PaparazziShowkaseLayoutDirectionProvider::class)
        direction: LayoutDirection,
    @TestParameter(valuesProvider = PaparazziShowkaseUIModeProvider::class)
        uiMode: PaparazziShowkaseUIMode,
  ): Unit {
    paparazzi.unsafeUpdateConfig(config.deviceConfig.copy(softButtons = false))
    takePaparazziSnapshot(paparazzi, elementPreview, direction, uiMode)
  }

  private object PaparazziShowkasePreviewProvider :
TestParameter.TestParameterValuesProvider {
```

```
    @TestParameter(valuesProvider = PaparazziShowkaseUIModeProvider::class)
        uiMode: PaparazziShowkaseUIMode,
  ): Unit {
    paparazzi.unsafeUpdateConfig(config.deviceConfig.copy(softButtons = false))
    takePaparazziSnapshot(paparazzi, elementPreview, direction, uiMode)
  }

  private object PaparazziShowkasePreviewProvider :
TestParameter.TestParameterValuesProvider {
    override fun provideValues(): List<PaparazziShowkaseTestPreview> {
      val metadata = Showkase.getMetadata()
      val components = metadata.componentList.map(::ComponentTestPreview)
      val colors = metadata.colorList.map(::ColorTestPreview)
      val typography = metadata.typographyList.map(::TypographyTestPreview)
      return components + colors + typography
    }
  }

  private object PaparazziShowkaseDeviceConfigProvider :
TestParameter.TestParameterValuesProvider {
    override fun provideValues(): List<PaparazziShowkaseDeviceConfig> = deviceConfigs()
  }
```

Wouldn’t it be amazing if I didn’t have to write all this code that you showed me today 🧘‍♀️

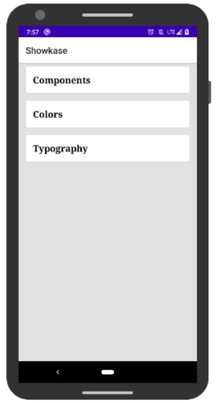
Showkase

https://github.com/**airbnb/Showkase**

Showkase

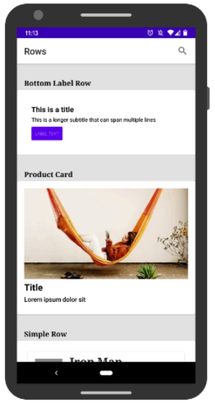
Showkase is an annotation-processor based Android library that helps you organize, discover, search and visualize Jetpack Compose UI elements. With minimal configuration it generates a UI browser that helps you easily find your components, colors & typography. It also renders your components in common situations like dark mode, right-to-left layouts, and scaled fonts which help in finding issues early.

Showkase auto-generates a browser for your Jetpack Compose UI Elements



All your UI elements are neatly organized automatically with minimal effort.

You can search your UI elements by their name or the group they belong to.

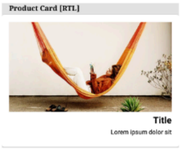


Get a preview of what each component in your codebase looks like. This helps with discoverability and better reuse.

Multiple permutations are auto generated for each component allowing you to preview it in different scenarios

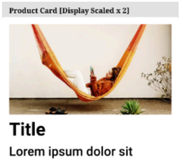
Basic Example

Dark mode enabled

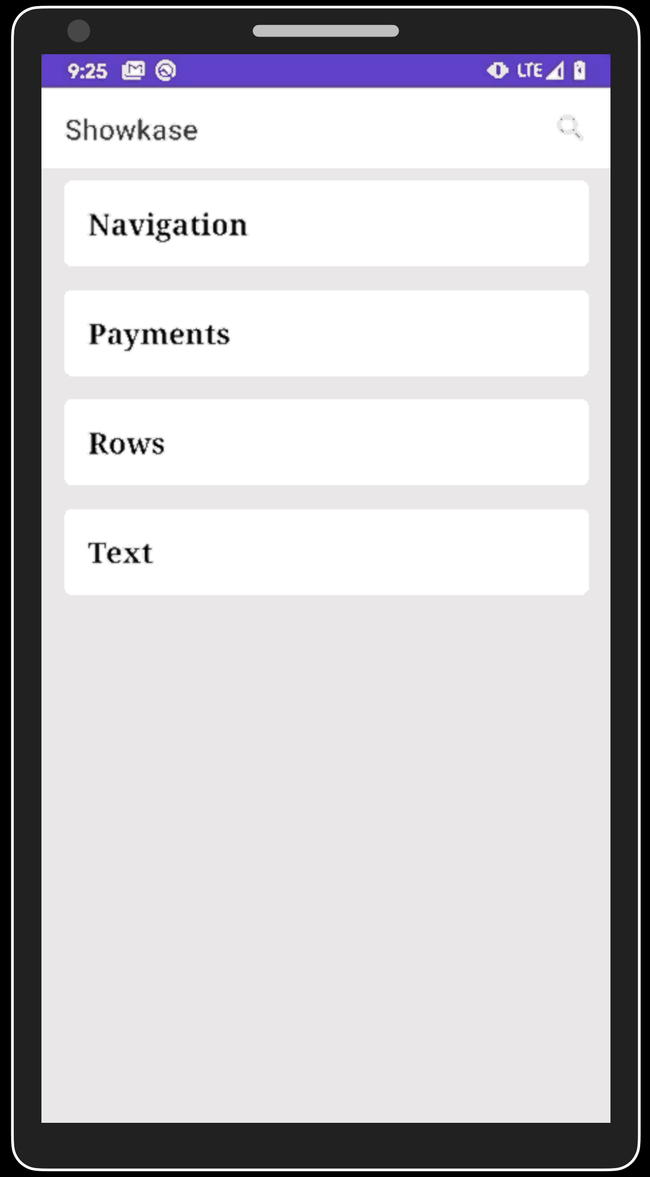


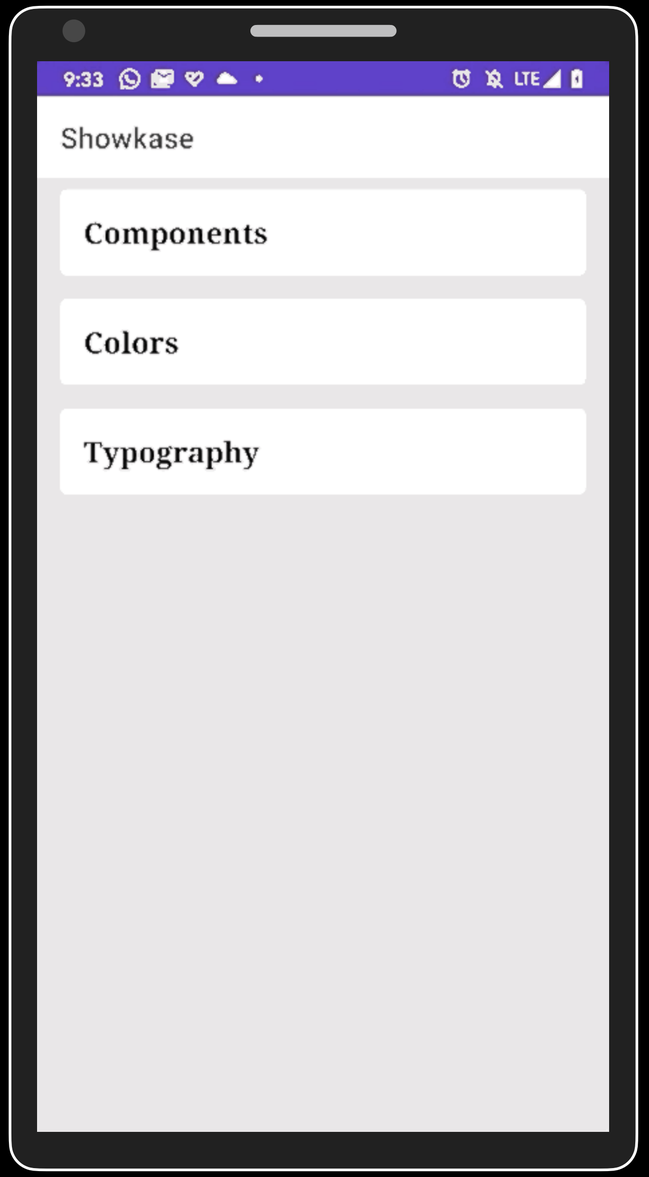
Right-to-left enabled

Font size is scaled to 2X the normal size



Density is scaled to 2X the normal size


9:25
LTE
Showkase
Navigation
Payments
Rows
Text


9:33
LTE
Showkase
Components
Colors
Typography

Summary

🔨 KSP needs to be in every engineer’s arsenal

🧩 Keep composability in mind when building UI infrastructure

😉 Don’t reinvent the wheel, use Showkase

Thank you, and don’t forget to vote

@VinayGaba

KotlinConf’23

Amsterdam