Compose Multiplatform

UI sharing einfach gemacht

Daniel Bälz & Robert Zetzsche

GDG DevFest Karlsruhe 2023

Daniel Bälz

Freiberuflicher Android Entwickler

https://dbaelz.de

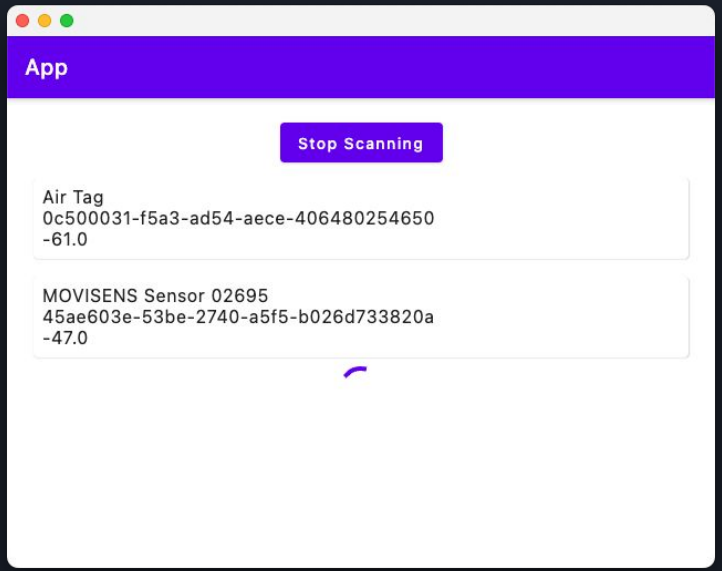
Robert Zetzsche

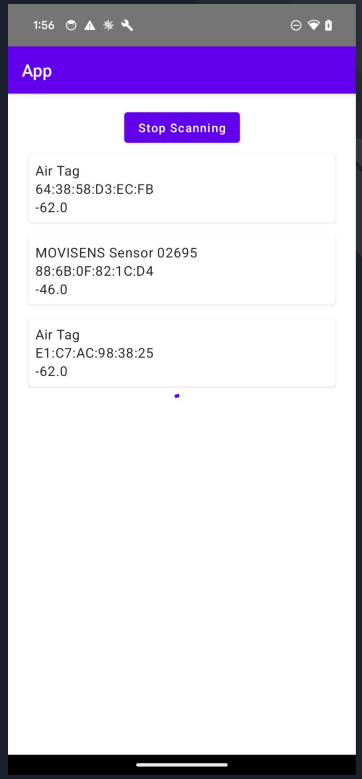
Senior Software Developer

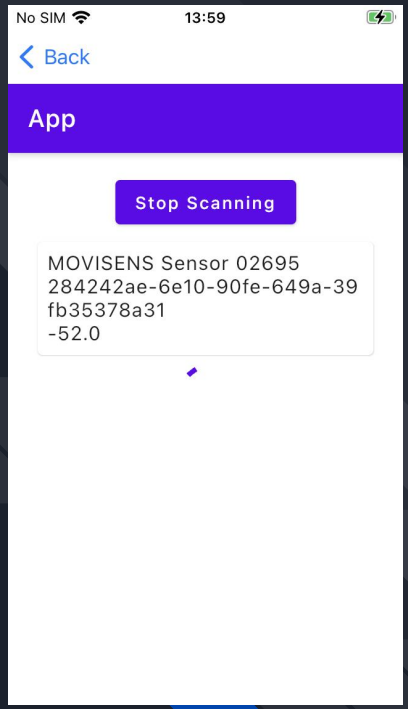
https://movisens.com

WERBUNG: Mobile Development Karlsruhe

https://www.meetup.com/de-DE/karlsruhe-mobile-development-meetup/


App
Stop Scanning
Air Tag
0c500031-f5a3-ad54-aece-406480254650
-61.0
MOVISENS Sensor 02695
45ae603e-53be-2740-a5f5-b026d733820a
-47.0


1:56
App
Stop Scanning
Air Tag
64:38:58:D3:EC:FB
-62.0
MOVISENS Sensor 02695
88:6B:0F:82:1C:D4
-46.0
Air Tag
E1:C7:AC:98:38:25
-62.0


No SIM
13:59
Back
App
Stop Scanning
MOVISENS Sensor 02695
284242ae-6e10-90fe-649a-39
fb35378a31
-52.0

Compose Multiplatform

- Deklaratives UI Framework
- Ziel: UI plattformübergreifend teilen
- Aufbauend auf Compose Compiler und Compose Runtime
  - Ebenfalls genutzt in Jetpack Compose UI
- Entwickelt von JetBrains (und Google)

Historie

- 2019 - Jetpack Compose
- 2020 - Jetbrains Compose
  - Prerelease (Android, Desktop, Web via HTML)
- 2022 - Jetbrains Compose 1.0
- 2023 - Compose Multiplatform 1.4.0
  - iOS und Wasm Alpha
- Aktuelle Version 1.5.10
  - Desktop Stable

Compose Multiplatform

Compose & Kotlin Multiplatform

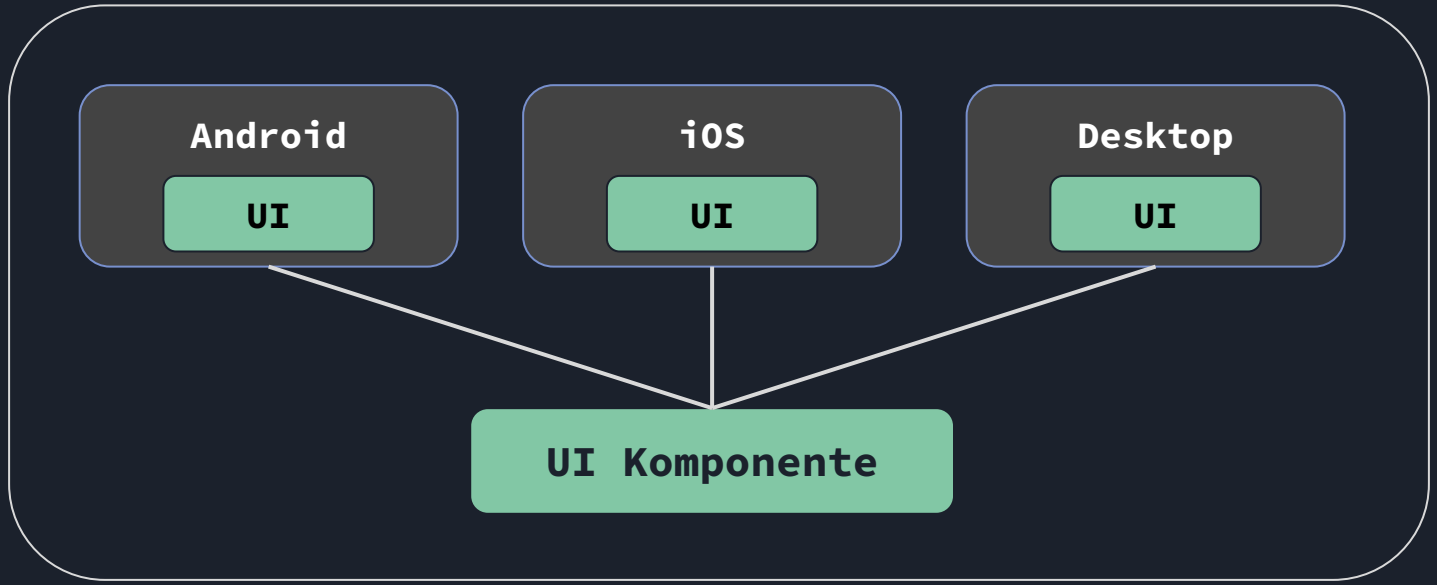
- Compose Multiplatform setzt auf Kotlin Multiplatform auf
  - Ermöglicht dadurch das Teilen von UI
  - Geteilt wird ebenfalls via Kotlin Multiplatform Artefakt
- Gleiche hierarchische Aufteilung
  - Common Code
  - Plattformspezifischer Code
    - Je Platform spezifische Composables
    - Interop und partielle Nutzung möglich

Compose Multiplatform Rendering

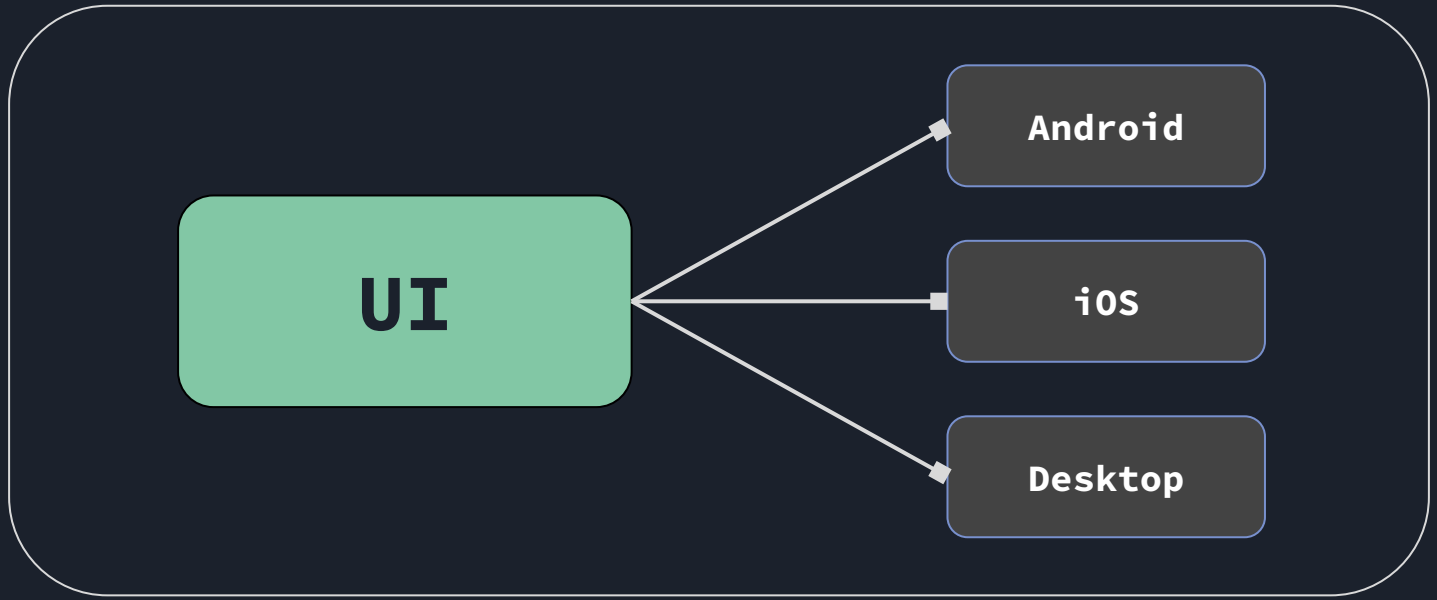
- Gerendert wird immer via Skia
- Android via Jetpack Compose UI
  - Alle Composables im plattformspezifischen Teil verfügbar
- iOS, Desktop, Web via Skiko

UI Architektur

Geteilte UI Komponenten



Geteilte UI



Imperative & deklarative UI

Beispiel imperative und deklarative UI

Abhängig vom Zahlenwert soll sich die Darstellung ändern

- 0: Schrift in Schwarz, Rahmen in Schwarz
- 1..5: Schrift in Grün, kein Rahmen
- 6..10: Schrift in Rot, kein Rahmen
- >= 11: Schrift in Rot, Rahmen in Rot

Imperative UI

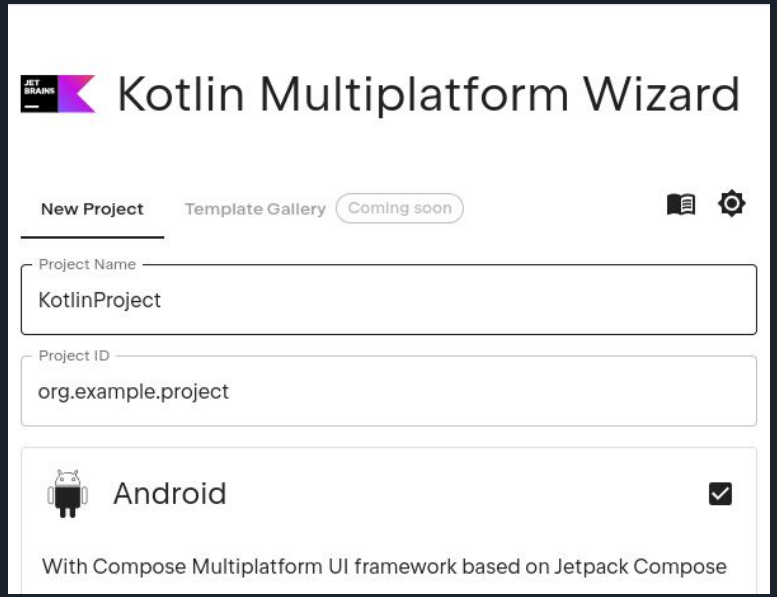
```
fun updateText(counter: Int) {
    setText(counter.toString())

    when (counter) {
        0 -> {
            setTextColor(Color.Black)
            setBorder(2.dp, Color.Black)
        }
        in 1..5 -> {
            setTextColor(Color.Green)
            removeBorder()
        }
        in 6..10 -> {
            setTextColor(Color.Red)
            removeBorder()
        }
        else -> {
            setTextColor(Color.Red)
            setBorder(2.dp, Color.Red)
        }
    }
}
```

Declarative UI

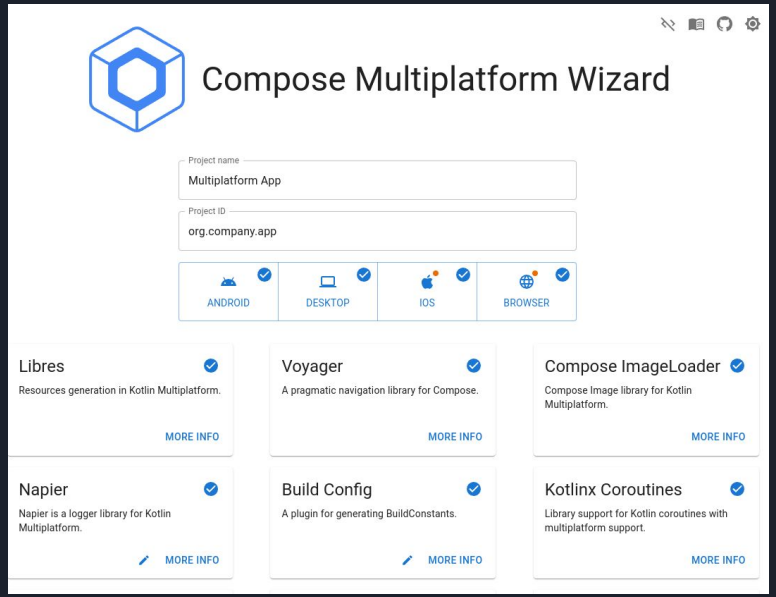
```
Text(
    text = counter.toString(),
    color = when (counter) {
        0 -> Color.Black
        in 1..5 -> Color.Green
        else -> Color.Red
    },
    modifier = when {
        counter == 0 -> Modifier.border(2.dp, Color.Black)
        counter >= 11 -> Modifier.border(2.dp, Color.Red)
        else -> Modifier
    }
)
```

Getting started

Neues Projekt starten



Offizielle KMP Wizard Webseite



Compose MP Wizard

...weitere Optionen

- Compose Multiplatform Template
- Android Studio/IDEA Projekt Templates*

- Neu: Amper
  - Experimentelles Tool zur Projektkonfiguration für IDEA und Fleet
  - Gradle Plugin mit YAML Konfiguration

* noch nicht auf Kotlin 1.9.20

Projektstruktur und Aufbau

- Kotlin Multiplatform Hierarchie
  - Common Code und Plattformspezifischer Code in Kotlin getrennt
  - Kotlin Multiplatform und Compose Multiplatform Code trennbar
- Core Compose Module im commonMain
  - bspw. `runtime`, `foundation` oder `material`
- Einige Compose Module die plattformspezifische UI Elemente liefern
  - `currentOs` oder `splitPane` bei Compose for Desktop
  - `core` oder `svg` bei Compose for HTML

Composable Function

Composable Function

```
@Composable
fun CounterButton() {
    var counter by remember { mutableStateOf(0) }
    Button(
        onClick = {  counter++ },
        modifier = Modifier.width(200.dp).height(50.dp)
    ) { Text("Counter: $counter") }
}
```

Integration

Interoperabilität mit anderen Frameworks

- **`expect`** und **`actual`**
  - Beta
  - Implementierung pro Plattform auf Kotlin Seite
  - Guter Ansatz um gezielt native UI Elemente zu verwenden

expect und **actual**

```
// commonMain
@Composable
expect fun NativeTextView(text: String)

// androidMain
@Composable
actual fun NativeTextView(text: String) = Text(text)

// iosMain
@Composable
actual fun NativeTextView(text: String) =
    UIKitView(modifier = Modifier, factory = { UILabel().also { it.text = text } })
```

Android

- Jetpack Compose UI
  - Material Design 2 und 3
  - Größte Funktionsumfang aller Compose Plattformen

- Interoperabilität
  - **`ComposeView`**: Compose innerhalb des Android View System
  - **`AndroidView`**: Android Views innerhalb Compose

iOS

- Compose in SwiftUI und in UIKit
  - Via `ComposeUIViewController`
- UIKit in Compose
  - Via `UIKitView`
- SwiftUI in Compose
  - Work in Progress
  - Muss in `UIView` gewrappt werden
  - KMP Limitation

iOS - Compose in SwiftUI und UIKit

```
// Kotlin
fun MainViewController(): UIViewController = ComposeUIViewController {
  Text("This is Compose code", fontSize = 20.sp)
}
// Swift
struct ComposeView: UIViewControllerRepresentable {
    func makeUIViewController(context: Context) -> UIViewController {
        Main_iosKt.MainViewController()
    }
    ...
}
```

iOS - UIKit in Compose

```
var message by remember { mutableStateOf(1) }
Button(onClick = { message += 1 }) { Text("Increment") }

UIKitView(modifier = Modifier, factory = {
    UILabel().also {
        it.text = message.toString()
        it.textColor = blackColor
    }
}, update = { it.text = message.toString() })
```

iOS - SwiftUI in Compose

```kotlin
// Kotlin
@OptIn(ExperimentalForeignApi::class)
fun MainViewController(
    createUIView: () -> UIView
) = ComposeUIViewController {
  MaterialTheme {
    Column {
      UIKitView(modifier = Modifier, factory = { createUIView() })
    }
  }
}
```

iOS - SwiftUI in Compose

```
class SwiftUIInUIView<Content: View>: UIView {
  init(content: Content) {
    super.init(frame: CGRect())
    let controller = UIHostingController
    controller.view.translatesAutoresizingMaskIntoConstraints = false
    addSubview(controller.view)
    NSLayoutConstraint.activate([
        controller.view.topAnchor.constraint(equalTo: topAnchor),
        controller.view.leadingAnchor.constraint(equalTo: leadingAnchor),
        controller.view.trailingAnchor.constraint(equalTo: trailingAnchor),
        controller.view.bottomAnchor.constraint(equalTo: bottomAnchor)
    ])
  }
}
```

iOS - SwiftUI in Compose

```
struct ComposeView: UIViewControllerRepresentable {
  func makeUIViewController(context: Context) -> UIViewController {
      Main_iosKt.MainViewController {
          SwiftUIInUIView(content: VStack{
              Text("TEST")
          })
      }
  }
    ...
}
```

Desktop Interoperabilität

- Swing
  - **`ComposePanel`**: Compose in Swing
  - **`SwingPanel`**: Swing innerhalb von Compose

- JavaFX
  - Keine Unterstützung von Compose in JavaFX
  - Rudimentäre Unterstützung in Compose mit **`JFXPanel`**

Desktop APIs

- Maus
  - Mouse Event listener, z.B. move, scroll und enter
  - Swing AWT events
  - Drag, Maustasten und umfangreichere click listener
- Keyboard
  - Focus events analog zu Android
  - Key events zum Beispiel für Sondertasten
- Context Menu, Tooltips, Tray Notifications, ...

Web

- Alt: Compose HTML mittels Kotlin/JS
  - Kein Compose Multiplatform
  - DSL um HTML/CSS zu erstellen
    - Kobweb wrappt dies um Compose ähnlichen Code zu schreiben
  - Maintenance Mode

- Neu: Compose Multiplatform for Web
  - Experimentelle Unterstützung basierend auf Kotlin/Wasm
  - Nutzt Feature wie WebAssembly GC
  - Viele KMP Bibliotheken unterstützen noch kein WebAssembly
  - Noch spezielle Compilerversion notwendig

Herausforderungen mit Compose

- Styling für spezifische Plattform schwierig
  - Material Design. Anderes Look’n’Feel muss nativ implementiert werden
  - Desktop: Aurora Framework
- Unterstützung von sehr unterschiedlichen Formfaktoren aufwendig
- Teilweise plattformspezifische Views sinnvoll

Bibliotheken und Ökosystem

Navigation

Voyager Decompose Router Circuit

Layouting

Window Size Class

Resources

MOKO Resources libres Compose ImageLoader

Roadmap

- Jetpack Compose core APIs und Komponenten werden schrittweise Multiplatform kompatibel
- Fokus: Compose for iOS Beta
  - Verbesserung der Rendering Performance
  - Scrolling und Textediting gleich zu nativen SwiftUI Apps
  - Accessibility Unterstützung
- Compose for Web Alpha
- Navigation direkt in Compose Multiplatform

Fazit

- Funktioniert in vielen Aspekten
- iOS und Web grundsätzlich nutzbar
  - Aber noch Alpha bzw. Experimental
- Desktop Stable
- Kotlin Multiplattform und Compose Multiplattform bieten große Flexibilität
  - Ermöglicht es Kotlin Entwicklern schnell verschiedene Plattformen zu bedienen
- Würden wir es schon einsetzen?

Bluetooth Kotlin Multiplatform

https://github.com/rzetzsche/BluetoothKotlinMultiplatform

Fragen?!