ToiletPaper #162

jambit WHERE INNOVATION WORKS

Jetpack Compose - Einfacher UIs für Android entwickeln

Autor: Mateusz Kalla / Software Architect / Media

✕ Problem

Um in Android ein User Interface zu bauen, werden für gewöhnlich XML-Layouts erstellt, die man dann im Java-/Kotlin-Code manipuliert. Damit ist es möglich, komplexe Nutzeroberflächen zu implementieren. Das bisherige Vorgehen hat jedoch auch seine Nachteile. Der Wechsel zwischen XML und Java/Kotlin ist nicht immer reibungslos – es fühlt sich natürlicher an, wenn man nur eine Sprache verwendet. XML ist außerdem nicht typsicher. Die IDE kann einen zwar dabei unterstützen, aber es ist beispielsweise immer noch möglich irrelevante und redundante Attribute hinzuzufügen. Außerdem sorgt XML für eine starke Kopplung zwischen UI und Logik: Nutzt man Funktionen wie findViewById, sind Kenntnisse darüber notwendig, wie eine XML-Datei aufgebaut ist. Mal abgesehen von den Nachteilen in XML ist teilweise viel Boilerplate-Code notwendig und die Lernkurve ist steil. Nehmen wir beispielsweise die Recyclerview: Man braucht einen Adapter, Viewholder, DiffUtil.ItemCallback, XML-Layouts etc.

✓ Lösung

Genau diese Probleme versucht die neue Android UI Library Jetpack Compose zu lösen. Man beschreibt seine UI komplett in Kotlin mit einem deklarativen Ansatz, statt wie bisher einem view-basierten. Obwohl Compose relativ neu ist (erste stabile Version Juli 2021), verfügt es jetzt schon über viele vorgefertigte Komponenten und sehr gute Dokumentationen. Es ist vollständig interoperabel. D. h. man kann auch nur Teile seines alten UI-Codes in Compose umschreiben. Dadurch, dass man Compose Code in Kotlin schreibt, wird durch den Scope und Typsicherheit sichergestellt, dass wirklich nur die relevanten Funktionen verfügbar sind. Außerdem kann man so viel leichter dynamische Views erzeugen, da man Zugriff auf die Kotlin-Operatoren hat. Compose bietet auch ein Preview-Feature mit dem man per Kotlin-Code Vorschauen in der IDE erzeugen kann, um seinen Code zu testen. So lassen sich z. B. unterschiedliche Vorschauen für Light- und Darkmode generieren. Ein weiterer Vorteil ist, dass man viel weniger Code benötigt, um seine UI zu implementieren – wie das nachfolgende Beispiel zeigt.

➔ Beispiel

Wie wenig Code in Compose nötig ist, sieht man am besten am Beispiel einer LazyColumn, dem Ersatz für die RecyclerView:

Unser Composable Code

```
@Composable
fun UserList(viewModel: UserViewModel) {

    val users = viewModel.users.collectAsState()

    LazyColumn(
        modifier = Modifier.fillMaxSize(),
        verticalArrangement = Arrangement.Center
    ) {
        items(users.value) { user ->
            User(user.name)
        }
    }
}


@Composable
private fun User(name: String) {
    Text(
        name,
        fontWeight = FontWeight.Bold,
        fontSize = 20.sp,
        modifier = Modifier.height(100.dp)
                           .padding(start = 10.dp)
                           .wrapContentSize()
    )
}
```

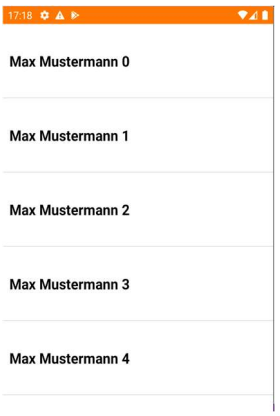
UI per setContent in der Activity

```
class MainActivity : AppCompatActivity() {

    ...
    override fun onCreate(savedInstanceState:
Bundle?) {
        super.onCreate(savedInstanceState)

        //Compose
        setContent {
            UserList(viewModel = userViewModel)
        }
    }
}
```

Ergebnis



Es ist kein Adapter, Viewholder, DiffUtil.ItemCallback oder XML-Layout notwendig.

✚ Weiterführende Aspekte

- https://medium.com/androiddevelopers/understanding-jetpack-compose-part-1-of-2-ca316fe39050
- https://medium.com/androiddevelopers/under-the-hood-of-jetpack-compose-part-2-of-2-37b2c20c6cdd
- https://medium.com/swlh/10-reasons-to-learn-jetpack-compose-ui-now-1fd6699f1f99