# Android Jetpack

```
@Composable
fun Authors() {
    Text("Tobias Fischer & Michael Zauner")
}
```

# Agenda

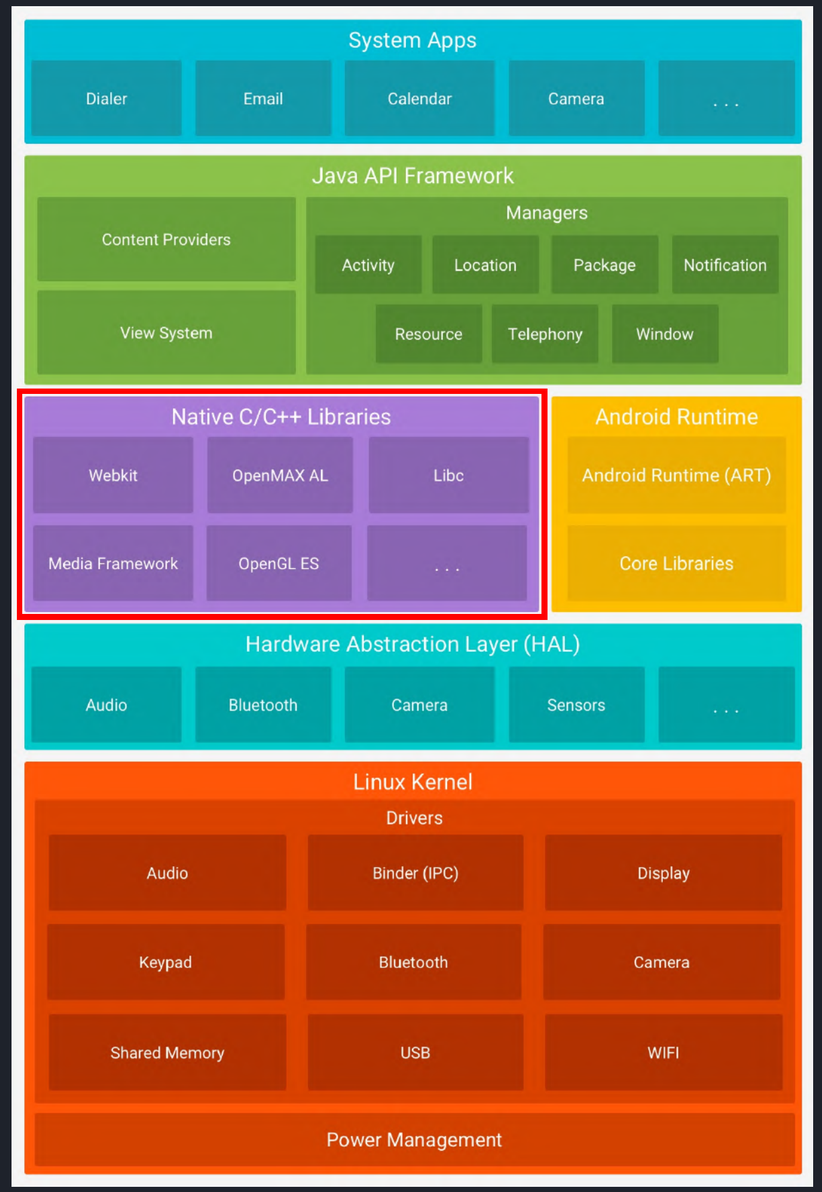
Android Libraries
Jetpack in General
Jetpack Compose
Room Database
Navigation
Workshop Blog App

# Android Libraries

NATIVE LIBRARIES

Funktionalitäten des Android-Runtime-Layers

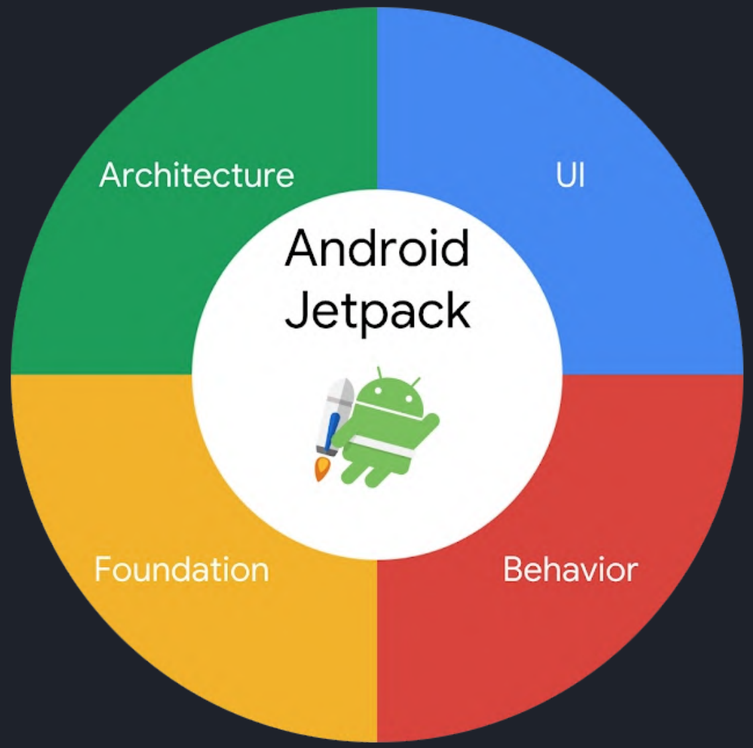
- SQLite: In-Memory relationale Datenbank
- WebKit: HTML, CSS und JavaScript
- OpenSSL: sichere Kommunikation mittels SSL/TLS
- ...



# Jetpack in General

"SUITE OF LIBRARIES FOR BETTER ANDROID DEVELOPMENT"

- vorgestellt @Google I/O 2018
- Sammlung nützlicher Android Libraries
  - aktuell fast 100
- Generelles Ziel: Boilerplate code minimieren
- Vorgehen: Libraries einzeln importieren
- abwärtskompatibel
  - unterstützt ältere Android-Versionen



# Jetpack Compose

"BUILD BETTER APPS FASTER"

UI-Framework

**Vorteile**

- weniger Code
- intuitiv
  - Code wird in Kotlin geschrieben
  - Android kümmert sich um den Rest
- verschnellert Entwicklung
- leistungsstark
- Verwendung neben bereits existierenden Code
- Test Integration

# Jetpack Compose

PARADIGMA DER DEKLARATIVEN PROGRAMMIERUNG

**Imperativer (alter) Ansatz**

- Baum von UI Widgets (**WIE** kommt man zum Ziel)
  - schrittweise DOM Manipulationen bis zum gewünschten Layout
    - ConstraintLayout → LinearLayout → TextView
- Zustandsänderungen manuell
  - `setText(String)`, `setVisibility(View.GONE)`, …
- fehleranfällig
  - Zustandsänderung kann fehlen
  - View wurde gelöscht
- XML-Layouts in Android

# Jetpack Compose

PARADIGMA DER DEKLARATIVEN PROGRAMMIERUNG

**Deklarativer (neuer) Ansatz**

- Layout und Logik verschmelzen in einer Datei
- gewünschtes Endergebnis bzw. Aussehen beschreiben (**WAS** will man am Ende)
  - keinen DOM Manipulationen
  - **Android baut Baum**
- Zustandsänderung: gesamten Bildschirm von Grund auf neu aufbauen
  - Screen komplett neu rendern
  - "**Recomposition**"
  - diesen Prozess versucht Compose möglichst intelligent umzusetzen
- Frameworks, welche diesen Ansatz benützen
  - Jetpack Compose, Flutter, SwiftUI, React, ...

# Jetpack Compose

## COMPOSABLE FUNCTIONS

- Basisfunktionen von Compose basieren auf Composable functions
- konvertieren Daten zu UI-Elementen
  - haben keinen Rückgabewert
- beschreiben Bildschirmzustand anstatt UI Widgets
- geschrieben in Kotlin
  - logische Konstrukte können verwendet werden (while, for, if, ...)
- annotiert mit `@Composable`

```
@Composable
fun Greeting() {
    Text("Hello World!")
}
```

# Jetpack Compose

## COMPOSABLE FUNCTIONS

weitere wichtige Fakten:

- können in jeglicher Reihenfolge und parallel ausgeführt werden
- Recomposition wird nur auf Objekte ausgeführt, welche sich verändern sollen
  - möglichst wenig neu rendern
- müssen sehr schnell sein
  - können unter Umständen sehr oft aufgerufen werden

können mittels anderer Argumente überschrieben werden:

```
@Composable
fun Greeting(name: String) {
    Text("Hello $name!")
}
```

# Jetpack Compose

## ZUSTÄNDE VERWALTEN

- einzige Weg um das UI upzudaten → erneuter Aufruf des Composables mit neuen Argumenten
- es wird Konstrukt benötigt, um:
  - aktuellen Zustand zu speichern
  - bei Zustandsänderung das Composable erneut aufzurufen

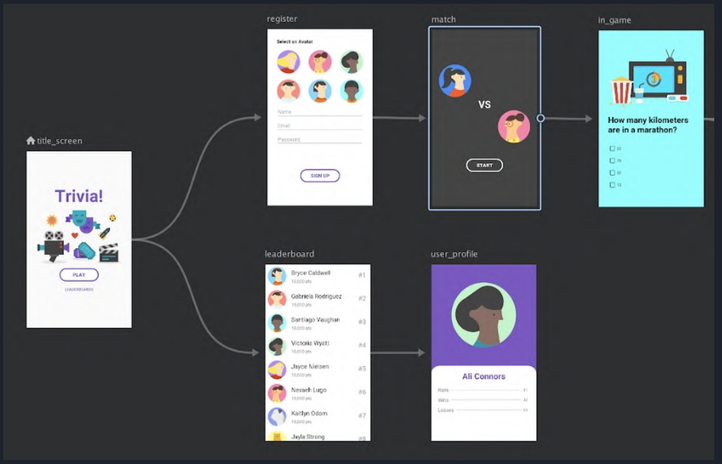
# Jetpack Compose

## ZUSTÄNDE VERWALTEN

- remember
  - sorgt dafür, dass eine Composable Function ein Objekt im Speicher halten kann
  - Composition (Objekt erzeugen): Wert wird gespeichert
  - Recomposition (Zustandsveränderung): Wert wird zurückgegeben
- mutableStateOf
  - plant Recomposition für alle Composeable Functions, welche diesen Wert verwenden

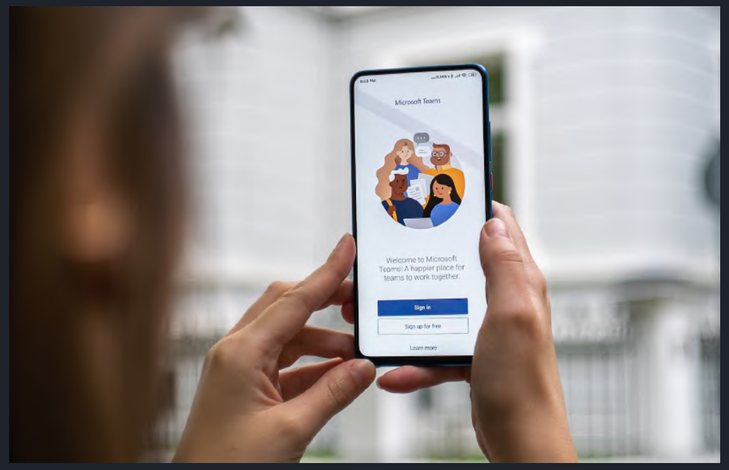
```
var name by remember { mutableStateOf("World") }

Greeting(name)
TextField(
    value = name,
    onValueChange = { name = it }
)
```

# Navigation




**NavGraph**

definiert Navigation Ziele




**NavHost**

Anzeige der unterschiedlichen Screens

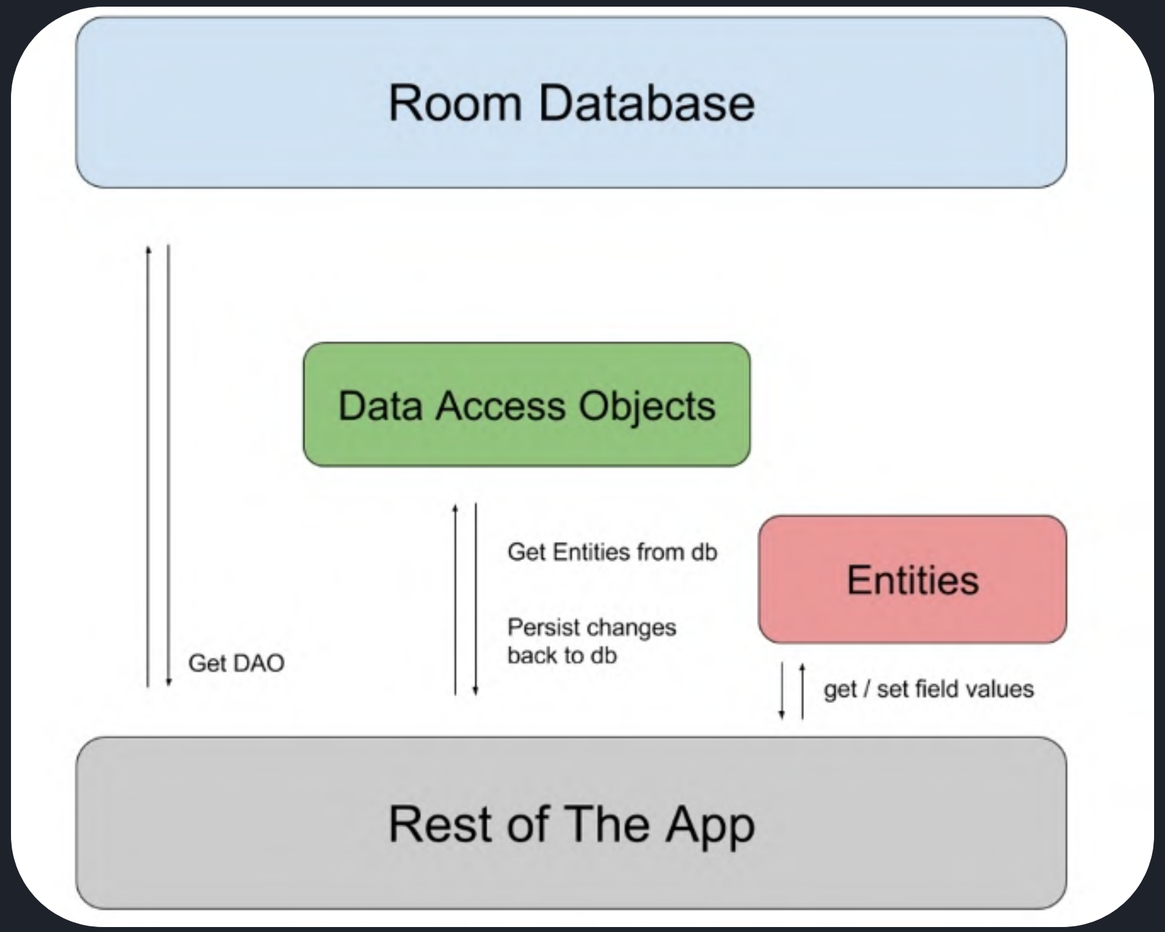
**NavController**

organisiert den Navigation Workflow

# Room

ABSTRAKTION VON SQLITE DB

- Validierung von SQL Queries bei Kompilierung
- Boilerplate Code minimieren mit Annotationen
- Unterstützung bei Migrationen (Datenmodell Änderung)



# Room - Entity

REFERENZ VON OBJEKT ZU EINTRAG IN DB

```
@Entity(tableName = "persons")
data class Person(
    @PrimaryKey val Id: Int,
    @ColumnInfo(name = "name") val fullName: String,
    @Ignore val picture: Bitmap
)
```

# Room - DAO

DATA ACCESS OBJECTS

```
@Dao
interface PersonDao{
    @Query("SELECT * FROM persons")
    fun getPersons(): Flow<List<Person>>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    fun insert(person: Person)
}
```

# Android Jetpack

```
@Composable
fun LetsCode() {
    Text("git clone https://github.com/tformatix/intro-android-jetpack.git")
}
```