Syntax Institut

Modul 3 – Android App Entwicklung mit Kotlin

Architektur und ViewModel

Gliederung

- Warum Architekturprinzipien?
- MVP und MVVM
- ViewModel
- Anwendung

Quelle: https://www.wien.info/de/sightseeing/sehenswuerdigkeiten/a-z/wotruba-366352

Warum Architekturprinzipien?

Eine gut gestaltete App ist:

- Besser zu **skalieren**
- Einfacher zu **erweitern**
- Besser zu **testen**
- Eignet sich für Arbeit in **Teams**

Quelle: https://www.gourmet-magazin.de/rezepte/spaghetti-knoblauch-pesto/

Populäre Architekturprinzipien

- Seperation of Concerns

  kleinste Komponenten mit wenig Verantwortung

  auch Fragment und Activity machen nur das notwendigste

- Drive UI from Model

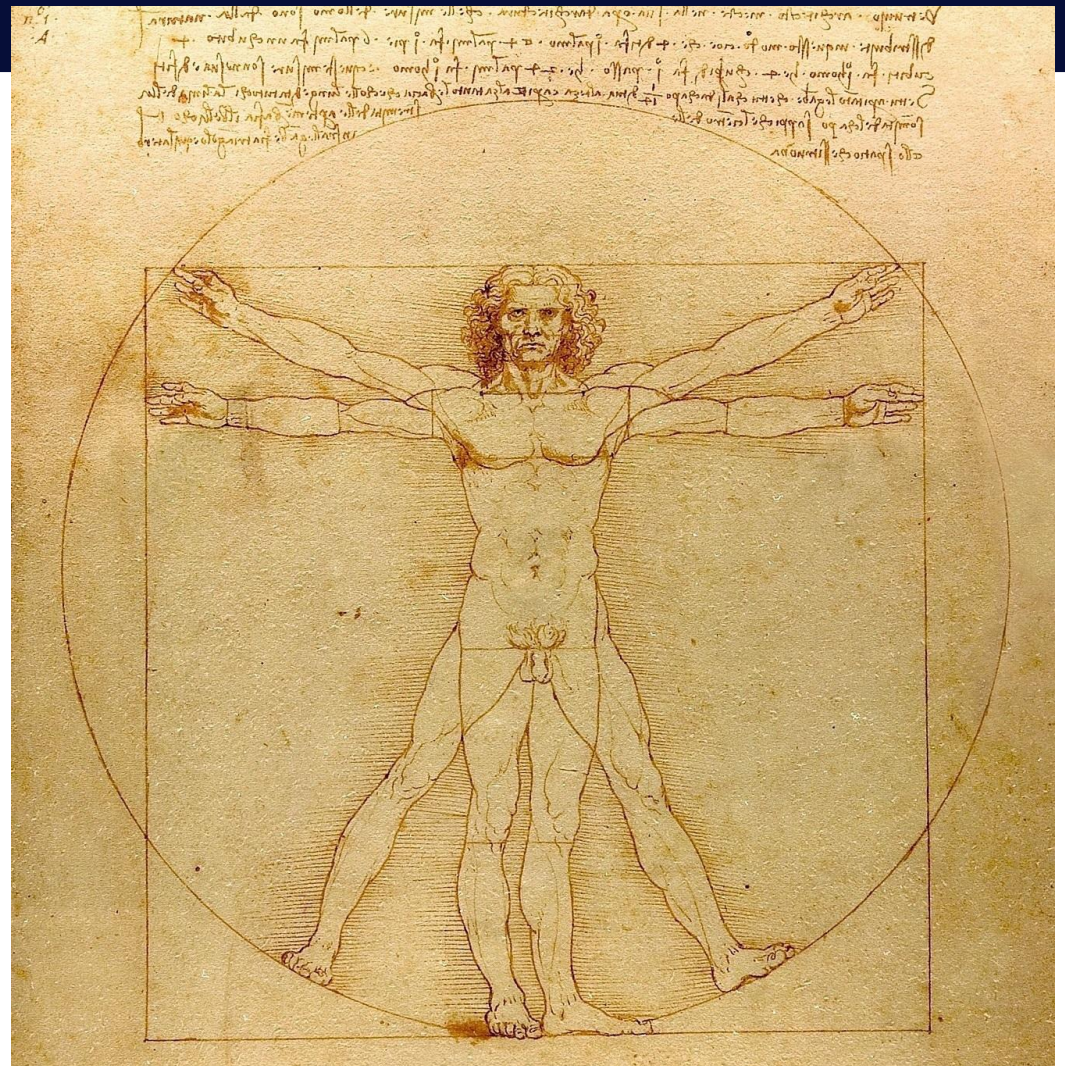
  UI wird von Lebenszyklus unabhängigen, anhaltenden Komponenten gesteuert

Quelle: https://www.schweizer-messer-shop.at/shop/schweizertaschenmesser/mittleretaschenmesser/handyman/

Architekturmuster

Gestaltungsvorlagen für „gute“ App Architektur

- Model View Controler - **MVC**
- Model View Presenter - **MVP**
- Model View ViewModel - **MVVM**



Quelle: https://www.whitewall.com/de/mag/goldener-schnitt

Model View ViewModel

von Google unterstützt und empfohlen

- Model

  Aufbereitung und Organisation der Daten

- View (Activities und Fragments)

  Verarbeitung unmittelbarer User Interaktion
  Anzeige von vom ViewModel bestimmten Daten

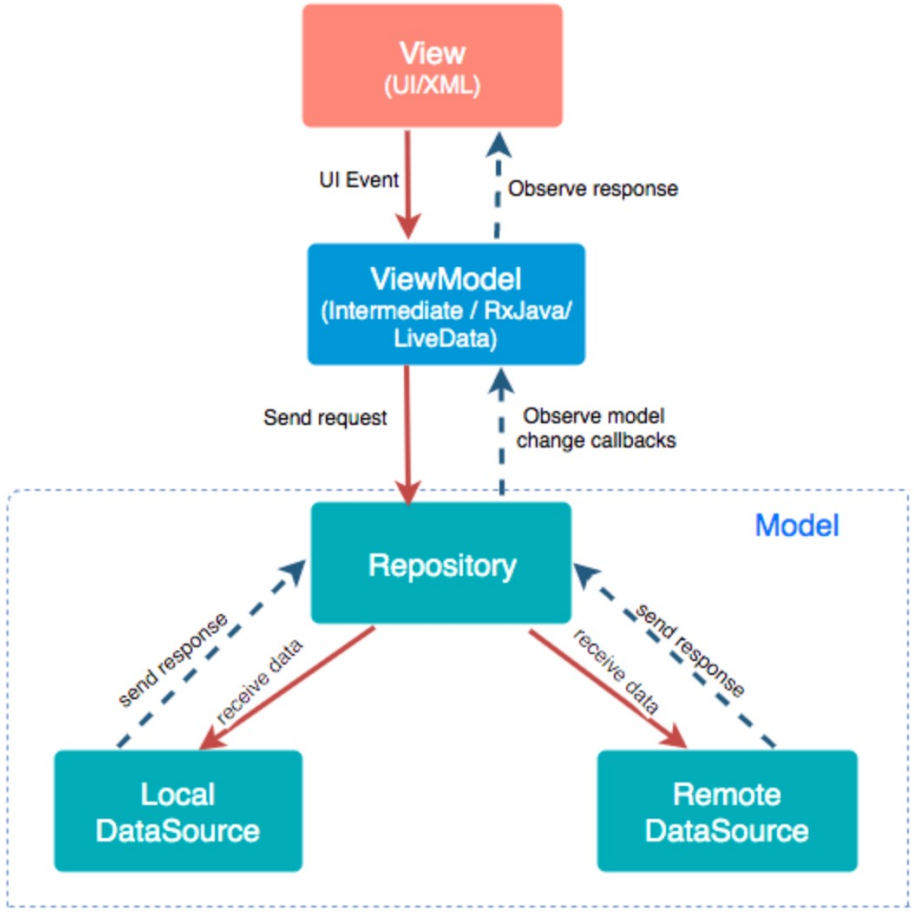
- ViewModel

  stellt Daten vom Repository bereit

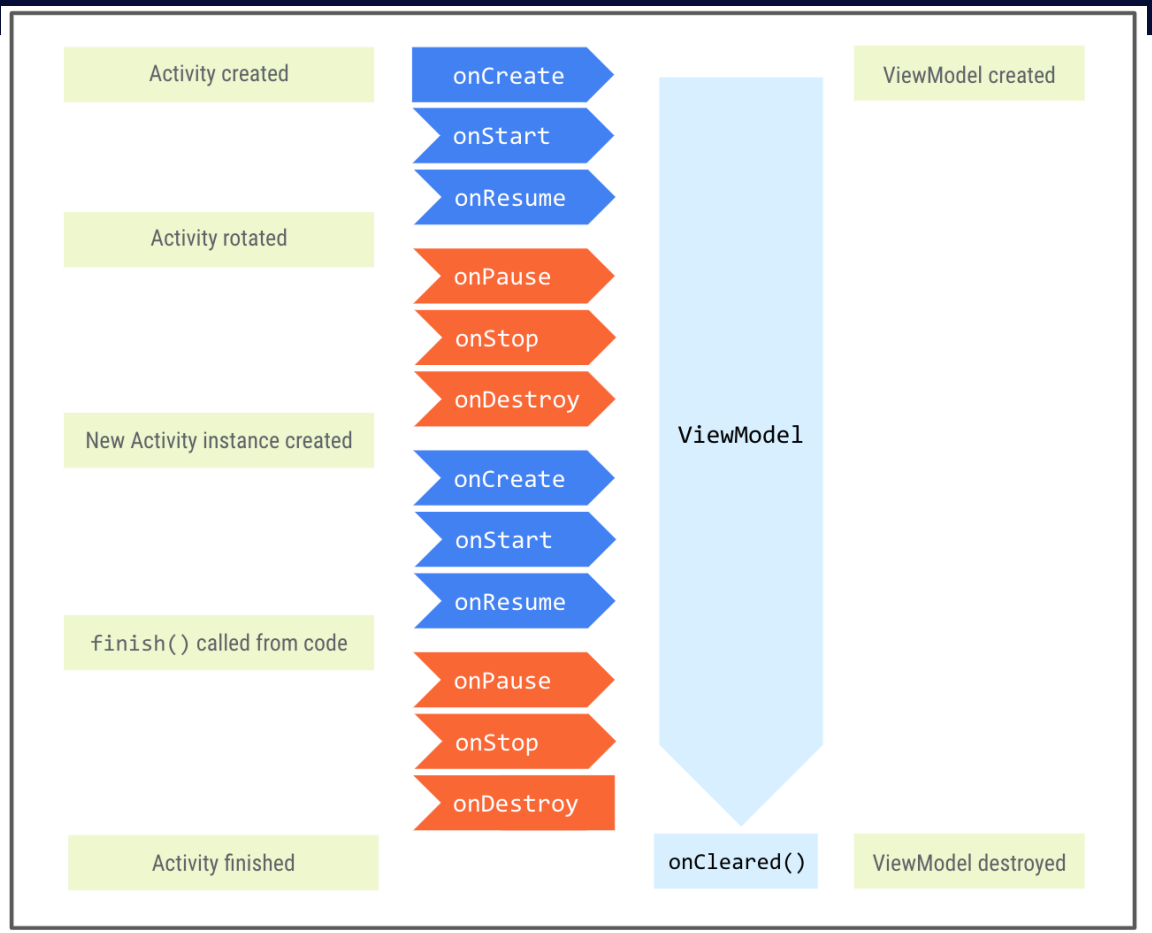
  enthält benötigte Logik

  wird von View beobachtet

  (meist via LiveData)



ViewModel Lifecycle



Quelle: https://developer.android.com/codelabs/basic-android-kotlin-training-viewmodel/img/18e67dc79f89d8a.png

Beispiel: ViewModel für QuizIt

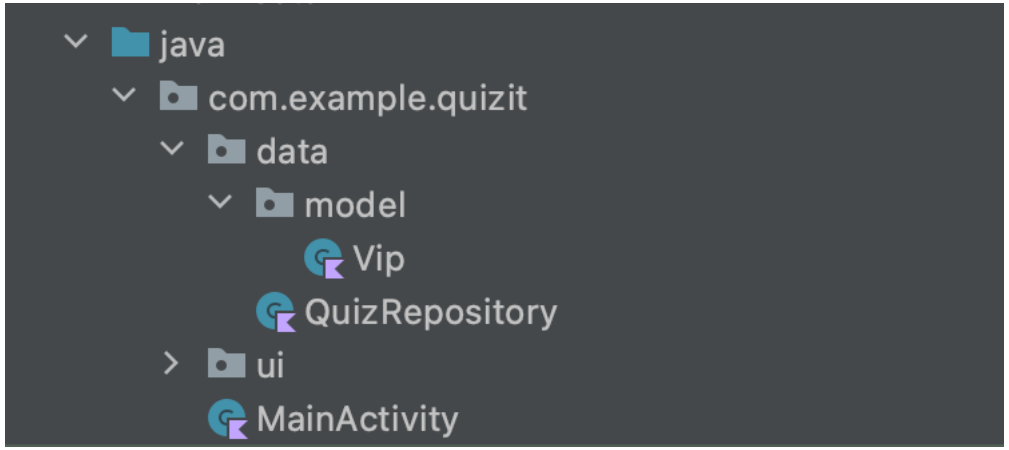
- **Repository** um Daten zu organisieren
- **QuizFragment** für UI und Anzeigen von Daten
- **QuizViewModel** für Spiellogik und zum Bereitstellen von Daten

Struktur

Im Hauptverzeichnis der App

- wird ein **data** Package angelegt dieses beinhaltet eine **QuizRepository** Klasse

- und in **data** wird ein weiteres Package namens **model** angelegt welches die **Vip** DatenKlasse beinhaltet



Vip.kt und QuizRepository.kt

die loadVips Funktion gibt eine Liste an VIPs zurück

VIPs haben einen **Namen** und **sind Musiker** (true)

oder Fußballer (false)

```kotlin
data class Vip(
    val name: String,
    val isMusician: Boolean
)
```

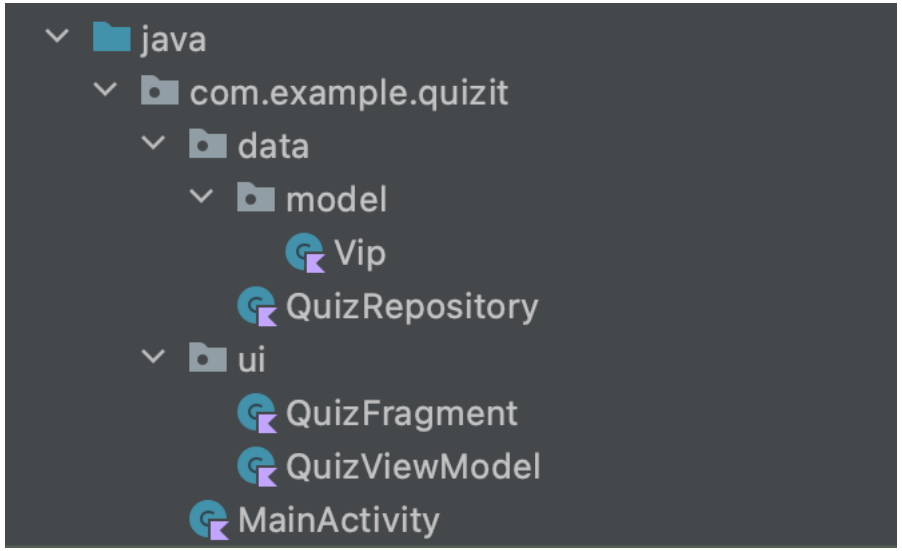
```kotlin
class QuizRepository {

    fun loadVips(): List<Vip> {
        val vipList = listOf(
            Vip( name: "Andrea Bocelli", isMusician: true),
            Vip( name: "Adriano Celentano", isMusician: true),
            Vip( name: "Eros Ramazotti", isMusician: true),
            Vip( name: "Antonio Vivaldi", isMusician: true),
            Vip( name: "Francesco Guccini", isMusician: true),
            Vip( name: "Franco Battiato", isMusician: true),
            Vip( name: "Lucio Battisti", isMusician: true),
            Vip( name: "Gino Paoli", isMusician: true),
            Vip( name: "Francesco Acerbi", isMusician: false),
            Vip( name: "Mario Balotelli", isMusician: false),
            Vip( name: "Andrea Belotti", isMusician: false),
            Vip( name: "Roberto Baggio", isMusician: false),
            Vip( name: "Alessandro Del Piero", isMusician: false),
            Vip( name: "Andrea Pirlo", isMusician: false),
            Vip( name: "Marco Materazzi", isMusician: false),
            Vip( name: "Gianluigi Buffon", isMusician: false)
        )

        return vipList.shuffled()
    }
}
```

Struktur

Im Hauptverzeichnis der App



legen wir ein weiteres Package namens **ui** an

welches

- **QuizFragment.kt** mit zugehörigem layout
- **QuizViewmodel.kt**

Beinhaltet.

activity_main.xml

Da unsere QuizApp aus einem Fragment besteht

Verzichten wir auf die Navigation Component und verweisen im Layout direkt auf das **QuizFragment**

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <androidx.fragment.app.FragmentContainerView
        android:id="@+id/fragmentContainerView"
        android:name="com.example.quizit.ui.QuizFragment"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

quiz_fragment.xml

Das layout des Fragments ist für DataBinding vorbereitet und besteht aus 3 TextViews und 2 Buttons

```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools">

    <data>

    </data>

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <TextView...>

        <TextView...>

        <Button...>

        <Button...>

        <TextView
            android:id="@+id/score_text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="50dp"
            android:textSize="50sp"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toBottomOf="@+id/title_text"
            tools:text="5" />
    </androidx.constraintlayout.widget.ConstraintLayout>
</layout>
```

QuizViewModel.kt

Das QuizViewModel lädt Daten vom Repository und kümmert sich um die Spiellogik

Folgende Verschachtelung:

```
private var _name = Wert
val name: Typ
get() = _name
```

stellt sicher dass Variablen nur innerhalb der Klasse verändert werden können

```
class QuizViewModel : ViewModel() {

    private val repository = QuizRepository()

    private val vipList = repository.loadVips()

    private var _score = 0
    val score: Int
        get() = _score

    private var _currentVip = vipList[0]
    val currentVip: Vip
        get() = _currentVip

    fun checkAnswer(guessMusician: Boolean) {

        if ((guessMusician && currentVip.isMusician) || (!guessMusician && !currentVip.isMusician)) {
            _score++
        }
        getNextVip()
    }

    private fun getNextVip() {
        val nextVip = vipList.random()

        if (nextVip == currentVip) {
            getNextVip()
        } else {
            _currentVip = nextVip
        }
    }

    fun restartGame() {
        _score = 0
        getNextVip()
    }
}
```

QuizFragment.kt

Das „schlanke“ Fragment kümmert sich nur noch um ViewBindings und ClickEvents

Jegliche weitere Funktionalität kommt vom ViewModel

dieses wird mittels

`private val viewmodel: QuizViewModel by viewModels()`

geladen

```
class QuizFragment : Fragment() {

    private val viewmodel: QuizViewModel by viewModels()

    private lateinit var binding: QuizFragmentBinding

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?,
    ): View? {...}

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        binding.vipNameText.text = viewmodel.currentVip.name
        binding.scoreText.text = viewmodel.score.toString()

        binding.footballButton.setOnClickListener {  it: View!
            checkAnswerUpdateUI( guessMusician: false)
        }

        binding.musicianButton.setOnClickListener {  it: View!
            checkAnswerUpdateUI( guessMusician: true)
        }
    }

    private fun checkAnswerUpdateUI(guessMusician: Boolean) {
        viewmodel.checkAnswer(guessMusician)
        binding.vipNameText.text = viewmodel.currentVip.name
        binding.scoreText.text = viewmodel.score.toString()

        if (viewmodel.score == 5) {
            showEndDialog()
        }
    }

    private fun showEndDialog() {...}

    private fun restartGame() {
        viewmodel.restartGame()
        binding.vipNameText.text = viewmodel.currentVip.name
        binding.scoreText.text = viewmodel.score.toString()
    }

    private fun exitGame() {
        activity?.finish()
    }
}
```

Zusatz: AlertDialog

Am Ende des Spiels wird ein **AlertDialog** gezeigt

Dieser benötigt

Title

Message

Negative Button inkl. ClickListener

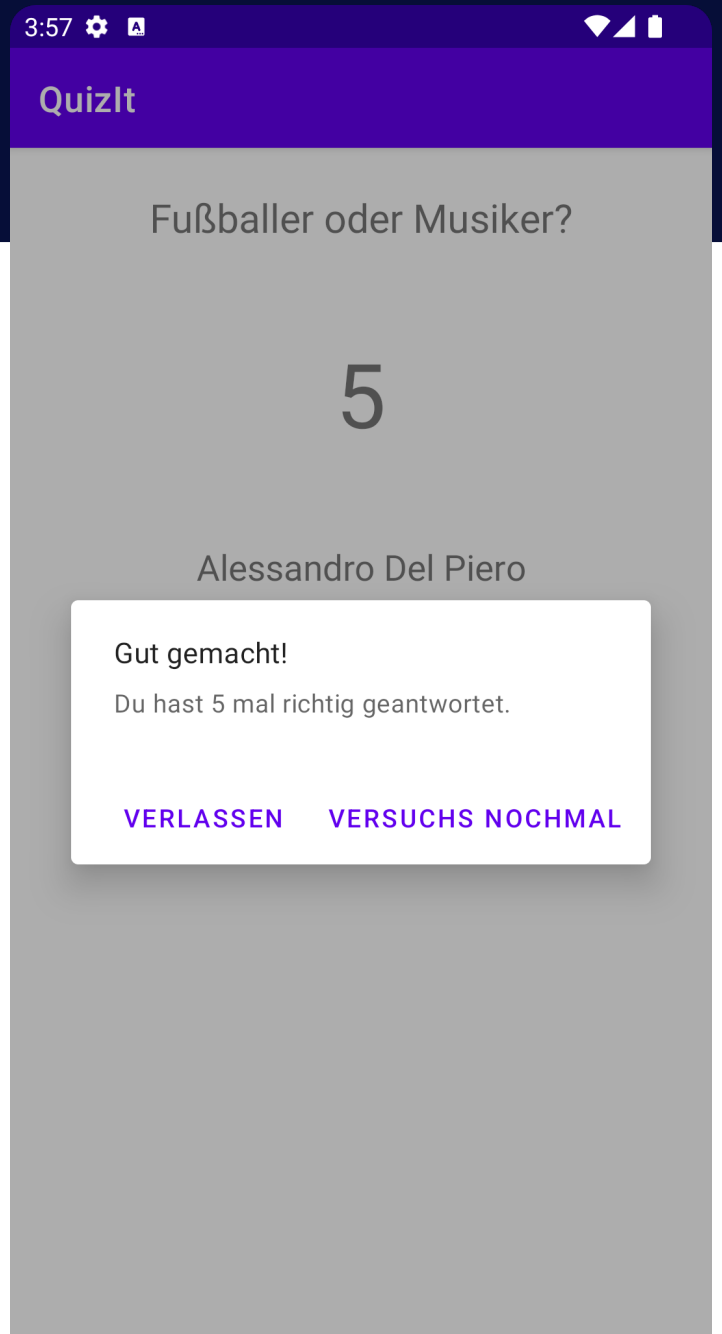
Positiv Button inkl. ClickListener

und

wird mittels **.show()** angezeigt

```
private fun showEndDialog() {
    MaterialAlertDialogBuilder(requireContext())
        .setTitle(getString(R.string.congratulations))
        .setMessage(getString(R.string.you_scored, viewmodel.score))
        .setCancelable(false)
        .setNegativeButton(getString(R.string.exit)) { _, _ ->
            exitGame()
        }
        .setPositiveButton(getString(R.string.play_again)) { _, _ ->
            restartGame()
        }
        .show()
}
```

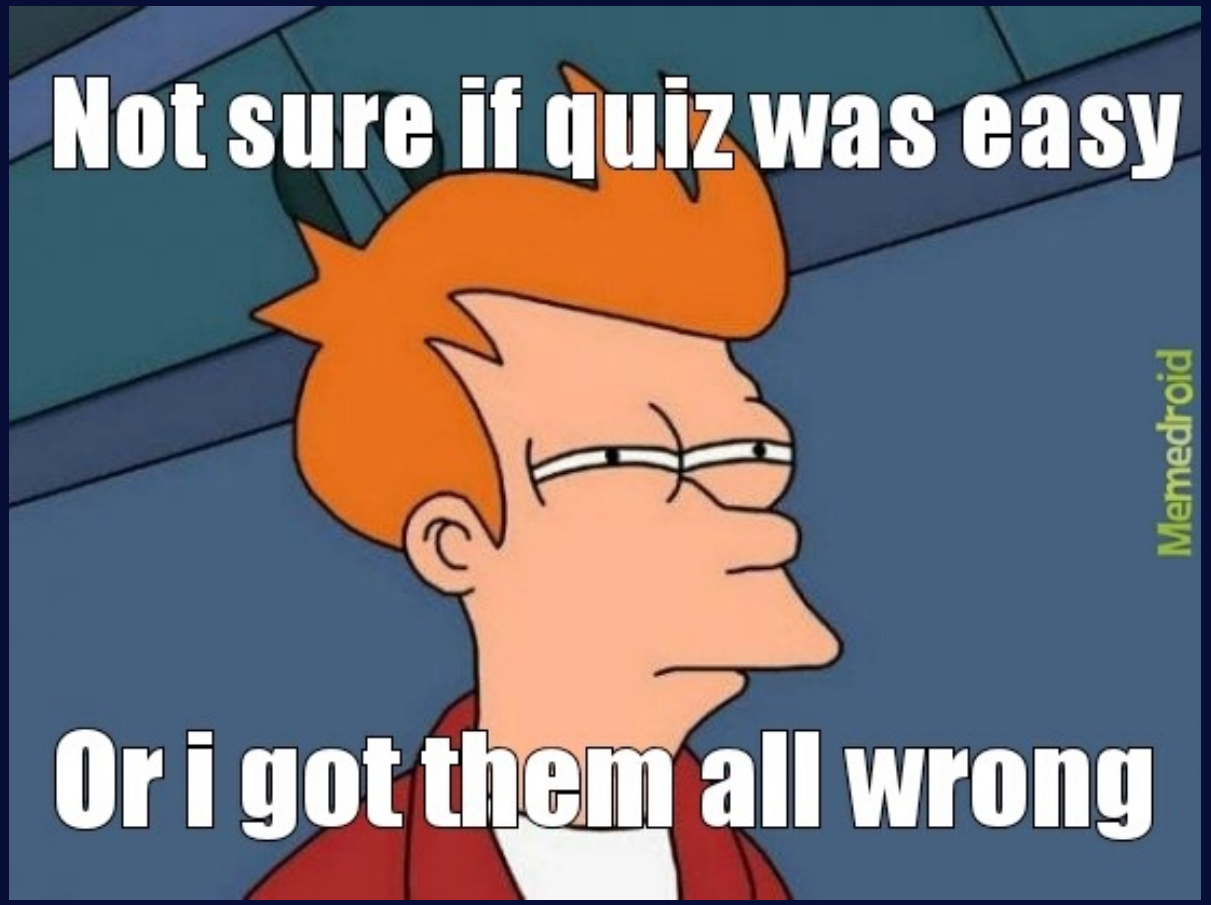
Fertig


3:57
QuizIt
Fußballer oder Musiker?
5
Alessandro Del Piero
Gut gemacht!
Du hast 5 mal richtig geantwortet.
VERLASSEN
VERSUCHS NOCHMAL

Architektur und ViewModel

Wiederholung - Was haben wir heute gelernt?

1	Architekturprinzipien
2	Model View ViewModel
3	ViewModel und Beispiel



Quelle: https://quizizz.com/admin/quiz/608ec579327f02001b132ed1/7d-wiederholungsquiz

Viel Spaß!